

FLASHCARDS

Java · Spring Boot · GCP · Microservicios

912 tarjetas de repaso rápido

frente = pregunta, dorso = respuesta · numeradas

★ **ESTRUCTURA = preguntas + chuleta de examen**
Java · 263 · Spring Boot · 231 · GCP · 212 · Microservicios · 206 · ANZADO.

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)

JAVA**J005**

¿Java pasa parámetros por valor o por referencia?

JAVA**J005**

RESPUESTA

SIEMPRE por valor.
Para objetos, el valor copiado es la referencia: el método puede mutar el objeto apuntado, pero reasignar el parámetro (param = otro) NO afecta al llamador.
Primitivos: copia directa, sin efecto externo.

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

Contenido: 912 tarjetas + chuleta exprés al dorso de esta portada · 230 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.

JAVA★

Sintaxis y tipos

J001

final / finally / finalize

BASICO

JAVA

Sintaxis y tipos

J002

¿Cuántos tipos de datos primitivos hay y cuáles son?

BASICO

JAVA

Sintaxis y tipos

J003

Primitivos vs wrappers (Integer, Double...)

BASICO

JAVA

Sintaxis y tipos

J004

¿Qué es el autoboxing y su trampa clásica?

BASICO

JAVA

Sintaxis y tipos

J005

¿Java pasa parámetros por valor o por referencia?

BASICO

JAVA

Sintaxis y tipos

J006

Overflow aritmético en int: ¿qué pasa?

BASICO

JAVA

Sintaxis y tipos

J007

Casting entre tipos numéricos: implícito vs explícito

BASICO

JAVA

Sintaxis y tipos

J008

¿Qué es var (Java 10+) y qué NO es?

BASICO

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

JAVA

Sintaxis y tipos

J002

RESPUESTA

8 primitivos: byte (8 bits), short (16), int (32), long (64), float (32), double (64), char (16, Unicode) y boolean.
Los demás tipos son objetos (clases, arrays, interfaces).

BASICO

JAVA



Sintaxis y tipos

J001

RESPUESTA

final: no se puede cambiar — variable (constante), método (no sobrescribible), clase (no heredable).
finally: bloque tras try/catch que casi siempre se ejecuta (limpieza).
finalize(): lo llama el GC antes de liberar el objeto; deprecado — no confíes en él.

BASICO

JAVA

Sintaxis y tipos

J004

RESPUESTA

Conversión automática primitiva $\leftrightarrow$ wrapper.
Trampa 1: un wrapper null al desempaquetar lanza NullPointerException.
Trampa 2: en loops grandes genera objetos (perf).
Trampa 3: comparaciones mixtas: `int == Integer` desempaqueta (NPE si null).

BASICO

JAVA

Sintaxis y tipos

J003

RESPUESTA

Primitivos: valor puro, rápido, no null, no van en colecciones.
Wrappers: objetos inmutables, permiten null, necesarios en genéricos/colecciones.
Autoboxing convierte automáticamente en ambas direcciones (con costo de rendimiento en loops).

BASICO

JAVA

Sintaxis y tipos

J006

RESPUESTA

Los enteros hacen wrap-around silencioso (sin excepción):
`Integer.MAX_VALUE + 1` $\rightarrow$ `Integer.MIN_VALUE`.
Soluciones: usar long, `Math.addExact/subtractExact` (lanzan `ArithmeticException`), o `BigInteger`.
Clásico bug: promedio $(a+b)/2 \rightarrow$ mejor `a + (b-a)/2`.

BASICO

JAVA

Sintaxis y tipos

J005

RESPUESTA

SIEMPRE por valor.
Para objetos, el valor copiado es la referencia: el método puede mutar el objeto apuntado, pero reasignar el parámetro (param = otro) NO afecta al llamador.
Primitivos: copia directa, sin efecto externo.

BASICO

JAVA

Sintaxis y tipos

J008

RESPUESTA

Inferencia de tipo local: `var lista = new ArrayList<String>()`; el tipo se fija en compilación (sigue siendo estático).
NO es tipo dinámico, NO aplica a campos, parámetros ni retornos, NO con null solo, NO sin inicializador.
Úsalo cuando el tipo es evidente a la derecha.

BASICO

JAVA

Sintaxis y tipos

J007

RESPUESTA

Implícito (widening): sin pérdida,
`byte` $\rightarrow$ `short` $\rightarrow$ `int` $\rightarrow$ `long` $\rightarrow$ `float` $\rightarrow$ `double` (`int` $\rightarrow$ `float` puede perder precisión en números grandes).
Explícito (narrowing): obligatorio y trunca: `int x=(int)3.99` $\rightarrow$ 3;
`byte b=(byte)130` $\rightarrow$ -126 (wrap).
`char` participa como numérico (0..65535).

BASICO

JAVA

Sintaxis y tipos

J009

Diferencia entre ++i e i++

BASICO

JAVA

Sintaxis y tipos

J010

¿Qué hace el operador ternario y su trampa con autoboxing?

BASICO

JAVA

Sintaxis y tipos

J011

switch clásico: ¿qué pasa sin break?

BASICO

JAVA

Sintaxis y tipos

J012

for vs while vs do-while vs for-each

BASICO

JAVA

Sintaxis y tipos

J013

Arrays en Java: características clave

BASICO

JAVA

Sintaxis y tipos

J014

¿Qué significa que un literal sea 1_000_000 o 0b1010?

BASICO

JAVA

Sintaxis y tipos

J015

¿Diferencia entre variable local, de instancia y estática?

BASICO

JAVA

Sintaxis y tipos

J016

¿Qué es instanceof y el pattern matching (Java 16+)?

BASICO

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

JAVA

Sintaxis y tipos

J010

RESPUESTA

condicion ? valorSi : valorNo.

Trampa: si un brazo es primitivo y otro wrapper, el resultado se unifica (puede desempaquetar y lanzar NPE con null):

Integer x = cond ? 1 : null; → ok

Integer y = cond ? 1 : objetoIntegerNull; → NPE al evaluar.

BASICO

JAVA

Sintaxis y tipos

J009

RESPUESTA

++i (pre-incremento): incrementa y devuelve el valor nuevo.

i++ (post-incremento): devuelve el valor viejo y luego incrementa.

En x = i++ se asigna el valor anterior. En expresiones complejas mezclarlos dificulta leer el código.

BASICO

JAVA

Sintaxis y tipos

J012

RESPUESTA

for: núcleo conocido de iteraciones.

while: repite mientras la condición sea true (puede no ejecutarse nunca).

do-while: ejecuta al menos una vez.

for-each (for tipo x : colección): recorre iterables sin índice; no permite modificar la estructura ni saltar elementos fácilmente.

BASICO

JAVA

Sintaxis y tipos

J011

RESPUESTA

Cae al siguiente caso (fall-through) y ejecuta todo lo demás hasta un break o el final.

El switch de expresión (Java 14+) con flecha (case X ->) NO cae: cada rama es independiente y requiere exhaustividad con enums/sealed.

BASICO

JAVA

Sintaxis y tipos

J014

RESPUESTA

Subrayados en literales numéricos solo separan dígitos (legibilidad), desde Java 7.

Prefijos: 0x hexadecimal, 0b binario, 0 octal (¡trampa!), sufijos L long, f float, d double.

'\u0041' es escape Unicode para 'A'.

BASICO

JAVA

Sintaxis y tipos

J013

RESPUESTA

Tamaño fijo, longitud con .length (sin paréntesis), índices 0..n-1, covariantes (String[] es Object[] → ArrayStoreException posible), inicialización: int[] a = {1,2,3};

Arrays.copyOf, Arrays.sort, Arrays.asList (vista de tamaño fijo).

Fuera de rango → ArrayIndexOutOfBoundsException.

BASICO

JAVA

Sintaxis y tipos

J016

RESPUESTA

instanceof verifica el tipo en runtime (null → siempre false).

Pattern matching: if (obj instanceof String s) { usar s } — hace el cast y declara la variable en una línea, sin repetir código.

BASICO

JAVA

Sintaxis y tipos

J015

RESPUESTA

Local: dentro de un método, vive durante la llamada, DEBE inicializarse antes de usarse (no hay default).

De instancia: por objeto, default según tipo (0, false, null).

Estática: por clase, compartida por todas las instancias, se inicializa al cargar la clase.

BASICO

JAVA

Sintaxis y tipos

J017

¿Cómo funciona el paso de String en un switch?

BASICO

JAVA

Sintaxis y tipos

J018

¿Qué es NaN y su trampa en comparaciones?

BASICO

JAVA

Sintaxis y tipos

J019

División entera vs decimal

BASICO

JAVA

Sintaxis y tipos

J020

¿Qué son los wrappers inmutables y cuáles tienen métodos útiles?

BASICO

JAVA

Sintaxis y tipos

J021

char vs String: diferencias clave

BASICO

JAVA

Sintaxis y tipos

J022

¿Qué es el alcance (scope) de una variable?

BASICO

JAVA

Sintaxis y tipos

J023

¿Cuándo usar long vs int vs double para dinero/tiempo?

BASICO

JAVA

Sintaxis y tipos

J024

Método main: firma válida y sobrecarga

BASICO

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

JAVA

Sintaxis y tipos

J018

RESPUESTA

Double.NaN es 'no es un número' (0/0.0).
NaN == NaN es FALSE; usar Double.isNaN() para detectarlo.
NaN en ordenamientos/comparaciones puede romper contratos de Comparator (Double.compare sí lo maneja).

BASICO

JAVA

Sintaxis y tipos

J017

RESPUESTA

Desde Java 7. Compara usando equals (case-sensitive), no ==.
Null en el switch → NullPointerException (verificar antes).
Con enum, los case van SIN calificar el nombre (case LUNES, no case Dia.LUNES).

BASICO

JAVA

Sintaxis y tipos

J020

RESPUESTA

Integer, Long, etc. son inmutables: las operaciones crean nuevos objetos.
Útiles: Integer.parseInt/valueOf, MAX_VALUE/MIN_VALUE, compare, Integer.parseInt(s, radix), Integer.toBinaryString, Character.isDigit/isLetter, Double.parseDouble.

BASICO

JAVA

Sintaxis y tipos

J019

RESPUESTA

int/int trunca: $7/2 \rightarrow 3$ (no redondea, hacia cero).
% es el residuo: $7\%2 \rightarrow 1$ (funciona con negativos con signo del dividendo).
Para decimal: que un operando sea double ($7/2.0 \rightarrow 3.5$).
División entera por cero → ArithmeticException; con doubles → Infinity o NaN.

BASICO

JAVA

Sintaxis y tipos

J022

RESPUESTA

Bloque: entre { } donde se declara (locales).
Campo: toda la clase (instancia/estática).
Parámetro: todo el método.
Una variable local puede ocultar un campo (this.x para desambiguar).
Los bloques internos NO pueden redeclarar una local del bloque externo.

BASICO

JAVA

Sintaxis y tipos

J021

RESPUESTA

char es un primitivo UTF-16 de 16 bits con comillas simples ('A');
String es objeto inmutable con comillas dobles.
char + int hace aritmética ('A'+1 → 66); char + String concatena.
Emojis y algunos Unicode ocupan 2 chars (surrogate pairs) → cuidado con length().

BASICO

JAVA

Sintaxis y tipos

J024

RESPUESTA

public static void main(String[] args) — también varargs String... args.
JVM solo invoca esa firma exacta; puedes SOBRECARGAR main pero la otra no se ejecuta sola.
args nunca es null (puede ser vacío), no incluye el nombre del programa.

BASICO

JAVA

Sintaxis y tipos

J023

RESPUESTA

Dinero: NUNCA double → BigDecimal o long de centavos.
Tiempo: epoch millis o nanos no caben en int (2038/overflow) → long.
Contadores pequeños y índices: int.
Identificadores grandes (IDs de DB): long.

BASICO

JAVA

Sintaxis y tipos

J025

varargs: sintaxis y reglas

BASICO

JAVA

Sintaxis y tipos

J026

¿Cómo se compila y ejecuta un programa Java? ¿Qué es el classpath?

BASICO

JAVA

Sintaxis y tipos

J027

Paquetes e imports: convención y propósito

BASICO

JAVA

Sintaxis y tipos

J028

Convenciones de nombres en Java (JavaBeans)

BASICO

JAVA

Sintaxis y tipos

J029

¿Qué es el Integer cache y cuándo sorprende?

INTERMEDIO

JAVA

Sintaxis y tipos

J030

BigDecimal: ¿cuándo y por qué?

INTERMEDIO

JAVA

Sintaxis y tipos

J031

¿Qué es 'effectively final' y por qué importa en lambdas?

INTERMEDIO

JAVA

Sintaxis y tipos

J032

Orden de inicialización de una clase

INTERMEDIO

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

JAVA

Sintaxis y tipos

J026

RESPUESTA

javac Hola.java → Hola.class (bytecode). java Hola → la JVM lo ejecuta.
 Varias clases/jars: java -cp "lib/*:clases" MiApp (classpath = dónde busca clases).
 Con Maven/Gradle esto lo automatiza el build; en un jar ejecutable: java -jar app.jar.
 -classpath mal puesto → ClassNotFoundException.

BASICO

JAVA

Sintaxis y tipos

J025

RESPUESTA

void sumar(int... nums) — acepta 0 o más ints (llega como array).
 Reglas: solo UNO por método y debe ser el ÚLTIMO parámetro.
 Sobrecarga con varargs es ambigua con facilidad: sumar(int...) y sumar(int, int...) → llamar sumar(5) no compila.
 Es azúcar sobre String[] args.

BASICO

JAVA

Sintaxis y tipos

J028

RESPUESTA

Clases/Interfaces/interfaces funcionales: PascalCase (MiServicio).
 Métodos y variables: camelCase (obtenerUsuario).
 Constantes: UPPER_SNAKE (MAX_INTENTOS).
 Paquetes: todo.minuscula.
 Getters/setters: getNombre/setNombre (isActive para boolean) — los frameworks (Jackson, Spring) confían en estas convenciones.

BASICO

JAVA

Sintaxis y tipos

J027

RESPUESTA

Paquete = namespace + organización: package com.empresa.proyecto.servicio;
 Convención: dominio invertido en minúsculas (com.google.x).
 import com.empresa.util.Fechas; o import estático: import static java.lang.Math.max;
 Sin import → nombre completo (java.util.List) o error de compilación.
 Los paquetes también definen el acceso package-private.

BASICO

JAVA

Sintaxis y tipos

J030

RESPUESTA

Para dinero y precisión decimal exacta: double/float usan binario (0.1+0.2 != 0.3).
 Crear SIEMPRE con String: new BigDecimal("0.1"), nunca new BigDecimal(0.1) (arrastra el error binario).
 Comparar con compareTo, no equals (equals considera escala: 1.0 != 1.00).
 Especificar scale y RoundingMode en divide().

INTERMEDIO

JAVA

Sintaxis y tipos

J029

RESPUESTA

Integer.valueOf() cachea de -128 a 127: dos autoboxed en ese rango son == el mismo objeto; fuera de él, objetos distintos.
 Por eso Integer a=127, b=127 → a==b es true, pero con 128 es false.
 Regla: en wrappers SIEMPRE usar equals(), nunca ==.

INTERMEDIO

JAVA

Sintaxis y tipos

J032

RESPUESTA

1) static: campos y bloques estáticos, en orden de aparición (una vez, al cargar).
 2) Por cada instancia: campos de instancia y bloques de instancia en orden, luego el constructor (que llama implícitamente a super() primero).
 Clásico de examen: el constructor de la clase padre corre ANTES que el cuerpo del hijo, pero los campos con inicializador del padre antes de su constructor.

INTERMEDIO

JAVA

Sintaxis y tipos

J031

RESPUESTA

Una variable local que no cambia después de inicializarse.
 Las lambdas y clases anónimas solo pueden capturar variables locales effectively final: se copian por valor al crear la lambda.
 Solución para acumular: usar un contenedor mutable (array de 1, AtomicInteger) o mejor refactorizar con reduce/collect.

INTERMEDIO

JAVA★

POO

J033

== vs equals() vs hashCode()

BASICO

JAVA★

POO

J034

¿Interfaz vs clase abstracta?

BASICO

JAVA★

POO

J035

Overloading vs Overriding

BASICO

JAVA★

POO

J036

¿Qué hace static?

BASICO

JAVA

POO

J037

¿Cuáles son los 4 pilares de la POO y qué significa cada uno?

BASICO

JAVA

POO

J038

Composición vs herencia: ¿cuándo cada una?

BASICO

JAVA

POO

J039

Constructores: reglas esenciales

BASICO

JAVA

POO

J040

this y super: usos

BASICO

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

JAVA ★
RESPUESTA

POO

J034

Interfaz: contrato puro; métodos abstract + default/static (Java 8+); una clase implementa varias.
Clase abstracta: no instanciable; tiene estado (campos), constructores y código común; herencia simple.
Interfaz para capacidad ('puede...'); abstracta para compartir estado.

BASICO

JAVA ★
RESPUESTA

POO

J033

== compara referencias (mismo objeto en memoria).
equals() compara estado; se sobrescribe junto a hashCode.
Contrato: si a.equals(b), entonces a.hashCode() == b.hashCode().
Si no cumples el contrato, HashMap/HashSet se rompen.

BASICO

JAVA ★
RESPUESTA

POO

J036

El miembro pertenece a la CLASE, no a cada instancia: compartido por todos los objetos y usable sin instanciar.
Clase anidada static: no guarda referencia al exterior.
Bloque static: inicialización una sola vez al cargar la clase.

BASICO

JAVA ★
RESPUESTA

POO

J035

Overloading (sobrecarga): mismo nombre, distinta firma (parámetros); se resuelve en COMPILACIÓN.
Overriding (sobrescritura): redefinir un método heredado con la misma firma; se resuelve en EJECUCIÓN según el tipo real del objeto (dynamic dispatch → polimorfismo).

BASICO

JAVA
RESPUESTA

POO

J038

Regla: favorece composición.
Herencia solo con relación is-a REAL y jerarquías estables (LSP garantizado); reutiliza y sustituye tipos.
Composición (has-a): delega, flexible, fácil de testear, evita acoplarse a detalles del padre.
Herencia rompe encapsulación: los cambios del padre afectan a los hijos.

BASICO

JAVA
RESPUESTA

POO

J037

Encapsulación: ocultar estado interno (campos private + métodos de acceso) y exponer comportamiento.
Abstracción: modelar lo esencial, exponiendo QUÉ hace, no CÓMO (interfaces/abstract).
Herencia: reutilización is-a de código y tipo (extends).
Polimorfismo: un mismo mensaje, comportamientos según el tipo real (overriding + dynamic dispatch).

BASICO

JAVA
RESPUESTA

POO

J040

this: referencia al objeto actual — desambiguar campos (this.x = x), llamar otro constructor (this(...)), pasar el objeto actual.
super: acceder al miembro del padre oculto (super.metodo()) o llamar su constructor (super(...)).
En métodos static NO existe ni this ni super.

BASICO

JAVA
RESPUESTA

POO

J039

Mismo nombre que la clase, sin tipo de retorno.
Si no defines ninguno, hay uno default sin args (desaparece al definir uno con args).
this(...) llama a otro constructor (debe ser la primera línea);
super(...) al del padre (también primera línea, implícito si no se escribe).
Constructores no se heredan ni se sobrescriben (solo se sobrecargan).

BASICO

JAVA

POO

J041

Modificadores de acceso en Java

BASICO

JAVA

POO

J042

¿Puede una clase abstracta no tener métodos abstractos? ¿Y una interfaz tener estado?

BASICO

JAVA

POO

J043

Métodos default y static en interfaces (Java 8+)

BASICO

JAVA

POO

J044

¿Qué es una clase inmutable y cómo se escribe una?

BASICO

JAVA

POO

J045

Patrón Builder: ¿cuándo vale la pena?

BASICO

JAVA

POO

J046

Clases anidadas: static nested vs inner vs local vs anónima

BASICO

JAVA

POO

J047

¿Qué métodos define Object y cuáles conviene sobrescribir?

BASICO

JAVA★

POO

J048

Contrato correcto de equals()

BASICO

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

JAVA

POO

J042

RESPUESTA

Sí: una clase abstracta puede no tener métodos abstractos (solo evita instanciarse).
 Una interfaz NO puede tener estado (campos de instancia); sus 'campos' son public static final constantes.
 Ambas pueden tener métodos default/static (interfaz) o concretos (abstracta).

BASICO

JAVA

POO

J041

RESPUESTA

private: solo la clase.
 (package-private, sin modificador): solo el mismo paquete.
 protected: paquete + subclases (aunque estén en otro paquete).
 public: todos.
 Top-level: solo public o package-private.
 Clases deben ser lo más cerradas posible.

BASICO

JAVA

POO

J044

RESPUESTA

Su estado no cambia tras construirse.
 Receta: clase final, campos final private, sin setters, constructor que copia defensivamente colecciones/arrays mutables, getters que devuelven copias de lo mutable, no exponer this durante construcción.
 Beneficios: thread-safe gratis, cacheable, seguro como clave de HashMap.

BASICO

JAVA

POO

J043

RESPUESTA

default: implementación en la interfaz, para evolución de APIs y mixins; la clase puede sobrescribirlo; en conflicto de diamante la clase DEBE resolverlo: InterfazA.super.metodo().
 static: método de la interfaz (Interfaz.metodo()), no se hereda.
 private (Java 9+): helpers internos compartidos entre defaults.

BASICO

JAVA

POO

J046

RESPUESTA

static nested: no necesita instancia externa (no guarda referencia al exterior).
 inner (no static): requiere instancia externa, accede a sus campos (Outer.this); puede causar memory leaks (actividad Android).
 local: dentro de un método.
 anónima: sin nombre, se implementa al vuelo (hoy en día suele reemplazarse por lambdas si es una interfaz funcional).

BASICO

JAVA

POO

J045

RESPUESTA

Cuando un constructor tendría muchos parámetros (especialmente opcionales del mismo tipo → errores de orden) o pasos opcionales.
 new Pizza.Builder("m").queso(true).aceitunas().build();
 Legible, inmutable al final, validación centralizada en build().
 Alternativa moderna: records + builder propio o withers.

BASICO

JAVA ★

POO

J048

RESPUESTA

Reflexivo: x.equals(x) true.
 Simétrico: x.equals(y) → y.equals(x).
 Transitivo: x=y, y=z → x=z.
 Consistente: mismo resultado en llamadas repetidas.
 x.equals(null) → false (nunca NPE).
 Receta: instanceof o getClass, comparar campos relevantes, usar Objects.equals para null-safe.

BASICO

JAVA

POO

J047

RESPUESTA

equals, hashCode, toString (casi siempre), clone (raro, mejor copias), getClass, notify/notifyAll/wait (final, concurrencia), finalize (deprecado).
 Regla de oro: equals y hashCode SIEMPRE juntos y consistentes.

BASICO

JAVA★

POO

J049

Contrato de hashCode() y qué pasa si lo rompes

BASICO

JAVA

POO

J050

toString(): buenas prácticas

BASICO

JAVA

POO

J051

Enums: capacidades que casi nadie usa

BASICO

JAVA

POO

J052

Clases sealed (Java 17+): ¿para qué sirven?

BASICO

JAVA

POO

J053

Records vs clases normales: ¿cuándo cada uno?

BASICO

JAVA

POO

J054

Comparable vs Comparator

BASICO

JAVA

POO

J055

Upcasting y downcasting

BASICO

JAVA

POO

J056

¿Cuándo marcar una clase/método final?

BASICO

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

JAVA

POO

J050

RESPUESTA

Generado por IDE/records para depurar rápido.
Incluir campos clave; NO incluir secretos ni datos personales (va a logs).
getClass().getSimpleName() + "(a=" + a + ")".
Evitar llamar métodos que disparan lazy loading (StackOverflow/exception en JPA).

BASICO

JAVA



POO

J049

RESPUESTA

Consistente entre llamadas; iguales por equals → mismo hashCode (obligatorio); desiguales → IDEALMENTE distinto (colisión permitida).
Si lo rompes: HashMap/HashSet almacenan en buckets distintos y no encuentran objetos 'iguales' → contains() false con un objeto que debería estar.
hashCode debe usar los MISMOS campos que equals.

BASICO

JAVA

POO

J052

RESPUESTA

Restringen QUIÉN puede heredar/implementar: public sealed interface Forma permits Circulo, Cuadrado.
Los permits deben ser final, sealed o non-sealed.
Habilitan switch con pattern matching exhaustivo (el compilador sabe todas las subclases).
Modela jerarquías cerradas de dominio con seguridad.

BASICO

JAVA

POO

J051

RESPUESTA

Constructor private, campos y métodos, puede implementar interfaces, abstract methods por constante (case por valor), valueOf/values(), ordinal (¡evita persistirlo!), compareTo por orden de declaración, EnumSet/EnumMap (rápidos).
Enum con constructor: enum Planeta { TIERRA(5.97){...} } con campos y comportamiento por constante.

BASICO

JAVA

POO

J054

RESPUESTA

Comparable (compareTo): orden NATURAL, lo implementa la propia clase (String, Integer).
Comparator: órdenes EXTERNOS y múltiples: Comparator.comparing(User::getNombre).thenComparing(User::getEdad).reversed().
Contrato: sgn(compare(x,y)) == -sgn(compare(y,x)), consistente con equals idealmente; romperlo rompe TreeSet/TreeMap.

BASICO

JAVA

POO

J053

RESPUESTA

Record: dato immutable transparente — genera constructor canónico, accesores sin 'get', equals/hashCode/toString; NO tiene setters, no hereda.
Clase: cuando necesitas mutabilidad, identidad con estado variable, herencia, o validación compleja post-construcción.
Records perfectos para DTOs, valores, resultados de cálculo y claves compuestas de Map.

BASICO

JAVA

POO

J056

RESPUESTA

Clase final: no heredable (String, Integer) — inmutabilidad y seguridad, diseños cerrados.
Método final: no sobrescribible — proteger invariantes.
Costo: pierde flexibilidad/testing con mocks de herencia. Diseña final por defecto en clases de valor; abierto solo donde hay extensión real pensada.

BASICO

JAVA

POO

J055

RESPUESTA

Upcasting (hijo→padre): implícito y seguro, pierde acceso a miembros del hijo.
Downcasting (padre→hijo): cast explícito, lanza ClassCastException si el tipo real no coincide; verificar con instanceof antes.
Downcasts frecuentes = diseño gritando por polimorfismo o genéricos.

BASICO

JAVA

POO

J057

Herencia de constructores y llamada implícita a super()

BASICO

JAVA

POO

J058

¿Qué es una interfaz funcional?

BASICO

JAVA

POO

J059

Diferencia entre abstracción e encapsulación (pregunta trampa)

BASICO

JAVA

POO

J060

¿Qué es el acoplamiento y la cohesión?
¿Cómo se relacionan con buen diseño?

BASICO

JAVA

POO

J061

Sobrecarga y resolución: ¿qué método se elige?

BASICO

JAVA

POO

J062

¿Se puede sobrecargar/sobrescribir main o un constructor?

BASICO

JAVA

POO

J063

¿Qué es y para qué sirve una clase de utilidades (utility class)?

BASICO

JAVA

POO

J064

¿Qué es el dynamic dispatch (enlace dinámico)?

INTERMEDIO

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

JAVA
RESPUESTA

POO

J058

Interfaz con EXACTAMENTE un método abstracto (SAM); puede tener defaults/statics.
Anotación @FunctionalInterface (opcional, la valida el compilador).
Destino de lambdas y referencias de método.
Core en java.util.function: Function<T,R>, Predicate<T>, Consumer<T>, Supplier<T>, BiFunction, UnaryOperator.

BASICO

JAVA
RESPUESTA

POO

J057

Los constructores NO se heredan.
Todo constructor empieza (implícita o explícitamente) con super() — si el padre NO tiene constructor sin args, el hijo DEBE llamar super(args) explícitamente o no compila.
El bloque del padre corre antes que el del hijo.

BASICO

JAVA
RESPUESTA

POO

J060

Cohesión: una clase con UNA responsabilidad clara (alta cohesión = bien).
Acoplamiento: cuánto depende una clase de otras (bajo acoplamiento = bien).
Objetivo: alta cohesión + bajo acoplamiento → fácil de cambiar, testear y reusar.
Herramientas: interfaces, inyección de dependencias, módulos.

BASICO

JAVA
RESPUESTA

POO

J059

Abstracción: decidir QUÉ expone el modelo (el contrato, el 'qué') — nivel de diseño.
Encapsulación: OCULTAR el estado y proteger invariantes (private + métodos) — mecanismo del lenguaje.
Se puede encapsular sin abstraer bien (getters de todo = encapsulación rota).

BASICO

JAVA
RESPUESTA

POO

J062

main: se puede SOBRECARGAR (otras firmas) pero la JVM solo ejecuta public static void main(String[]).
Constructores: solo SOBRECARGA (entre ellos con this(...)); no existe overriding de constructores porque no se heredan.
Un método static NO se sobreescribe (se oculta, hiding).

BASICO

JAVA
RESPUESTA

POO

J061

El compilador elige el método por el tipo ESTÁTICO de los argumentos, con la conversión más específica posible (primitivo exacto > widening > autoboxing > varargs).
Trampa: sum(5) con sum(int) y sum(long): elige int.
Con null ambiguo entre sobrecargas → error de compilación (hay que hacer cast).

BASICO

JAVA
RESPUESTA

POO

J064

En runtime se invoca el método del TIPO REAL del objeto, no del tipo de la referencia:
Animal a = new Perro(); a.hablar(); → ejecuta Perro.hablar().
Solo aplica a métodos de instancia (no static, no campos, no private/final).
Los CAMPOS no son polimórficos: se resuelven por el tipo de la referencia (field hiding).

INTERMEDIO

JAVA
RESPUESTA

POO

J063

Clase final con constructor private (no instanciable) y solo métodos static: Collections, Arrays, StringUtils.
Útil para funciones puras sin estado.
Crítica: dificulta mocking y polimorfismo; en código testeable se prefieren componentes inyectados con interfaz.

BASICO

JAVAJ065POO

¿Qué es el patrón Método Fábrica estático (effective Java item 1) y por qué preferirlo a constructores?

INTERMEDIO

JAVAJ066POO

Field hiding vs overriding: diferencia sutil pero de examen

INTERMEDIO

JAVAJ067POO

clone() y por qué casi siempre se evita

INTERMEDIO

JAVAJ068POO

¿Qué es la covarianza de arrays y su riesgo?

AVANZADO

JAVAJ069★String y texto

¿Por qué String es immutable?

BASICO

JAVAJ070String y texto

Métodos esenciales de String que hay que saberse

BASICO

JAVAJ071String y texto

StringBuilder vs StringBuffer vs String.concat

BASICO

JAVAJ072String y texto

split() y su trampa con espacios finales y caracteres especiales

BASICO

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

JAVA

POO

J066

RESPUESTA

Campos y métodos static se resuelven por el TIPO DE LA REFERENCIA (hiding, en compilación).
Métodos de instancia se resuelven por el TIPO REAL del objeto (overriding, en runtime).
Padre p = new Hijo(); p.x da el campo del Padre; p.metodo() ejecuta el del Hijo.

INTERMEDIO

JAVA

POO

J065

RESPUESTA

Métodos static que devuelven instancias: List.of(), Integer.valueOf(), Optional.of().
Ventajas: tienen NOMBRE, pueden cachear, devolver subtipos, no requieren crear objeto nuevo.
Convenciones: of, ofNullable, valueOf, from, newInstance.

INTERMEDIO

JAVA

POO

J068

RESPUESTA

String[] es subtipo de Object[] (arrays covariantes en runtime).
Riesgo: Object[] arr = new String[1]; arr[0] = 42 compila pero lanza ArrayStoreException en runtime.
Las colecciones NO son covariantes (List<String> no es List<Object>) — por eso existen los wildcards.

AVANZADO

JAVA

POO

J067

RESPUESTA

Object.clone es protected, requiere implementar Cloneable (que no tiene métodos) y hacer super.clone(); con campos mutables hay que hacer copia profunda a mano.
Alternativas: constructor de copia (new Punto(Punto otro)), método estático copiaDe(), o record (immutable no necesita clonar).
Effective Java: prefiere factories de copia.

INTERMEDIO

JAVA

String y texto

J070

RESPUESTA

length, charAt, substring(begin[,end]) — end exclusivo, indexOf/lastIndexOf, contains, startsWith/endsWith, replace, split, trim/strip (strip es Unicode, Java 11), toUpper/LowerCase, isEmpty/isBlank (sin contenido o solo espacios), equals/equalIgnoreCase, compareTo, chars(), repeat, join, format.

BASICO

JAVA



String y texto

J069

RESPUESTA

Su contenido no cambia tras crearse (los métodos devuelven nuevos Strings).
Beneficios:

- String pool reutiliza instancias
- Thread-safe
- hashCode cacheado
- Seguro como clave en HashMap y para rutas/URLs

BASICO

JAVA

String y texto

J072

RESPUESTA

split usa regex: split(".") NO funciona (punto = 'cualquier carácter') → escapar split("\\.").
Los strings vacíos del FINAL se descartan por defecto: "a,b,.".split(",") → [a,b].
Usar split(",", -1) para conservarlos (útil en CSV).

BASICO

JAVA

String y texto

J071

RESPUESTA

StringBuilder: mutable, NO sincronizado (el normal desde Java 5).
StringBuffer: igual pero sincronizado (legado).
concat/+ para pocas uniones.
Capacidad inicial 16; crece duplicando+2. new StringBuilder(tamañoEstimado) evita reallocs.

BASICO

JAVA

String y texto

J073

String.format y text blocks

BASICO

JAVA

String y texto

J074

trim() vs strip() vs isBlank()

BASICO

JAVA

String y texto

J075

Comparar Strings correctamente en todos los casos

BASICO

JAVA

String y texto

J076

String.join y StringJoiner

BASICO

JAVA

String y texto

J077

¿Cómo iterar los caracteres de un String moderno?

BASICO

JAVA

String y texto

J078

Conversión String ↔ número y sus excepciones

BASICO

JAVA

String y texto

J079

¿Qué es un text block y cómo maneja la indentación?

BASICO

JAVA

String y texto

J080

¿Qué es el String pool y cómo funciona?

INTERMEDIO

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

JAVA

String y texto

J074

RESPUESTA

trim() elimina caracteres <= espacio (ASCII, legado).
strip() (Java 11) usa Character.isWhitespace — Unicode correcto.
isBlank() true si vacío o solo whitespace.
isEmpty() solo true con length 0.

BASICO

JAVA

String y texto

J073

RESPUESTA

format("%s tiene %d años", nombre, edad): %s string, %d entero, %f decimal (%.2f 2 decimales), %n salto, %% literal.
Text blocks (Java 15+): """ ... """ multilinea; quita indentación común (incidental), respeta saltos; útiles para JSON/SQL legible.

BASICO

JAVA

String y texto

J076

RESPUESTA

String.join(", ", "a", "b") o String.join(delim, lista).
StringJoiner permite prefijo/sufijo: new StringJoiner(", ", "[", "]").
Equivalente en streams: Collectors.joining(", ", "[", "]").

BASICO

JAVA

String y texto

J075

RESPUESTA

SIEMPRE a.equals(b), no ==.
Ignorando mayúsculas: equalsIgnoreCase o mejor compareTo case-normalized.
Null-safe: Objects.equals(a,b) o "constante".equals(variable).
Orden natural: compareTo (0 = iguales) — cuidado con null (NPE).

BASICO

JAVA

String y texto

J078

RESPUESTA

Integer.parseInt("42") → int; valueOf("42") → Integer.
Double.parseDouble, Long.parseLong.
Formato inválido → NumberFormatException (checked? NO, unchecked).
Convertir número a String: String.valueOf(x) o Integer.toString(x).
Base distinta: parseInt("ff", 16) → 255.

BASICO

JAVA

String y texto

J077

RESPUESTA

chars() devuelve IntStream: "abc".chars().filter(c -> c != 'a')...
Loop clásico con charAt(i) para índices.
OJO: codePoints() para Unicode real (emojis ocupan 2 chars; chars() parte los surrogate pairs).

BASICO

JAVA

String y texto

J080

RESPUESTA

Área del heap donde la JVM guarda Strings únicos. Los literales ("hola") se internan automáticamente: dos literales iguales son el MISMO objeto.
new String("hola") crea objeto FUERA del pool (por eso == da false); .intern() lo mete/busca en el pool.
Ahorra memoria y permite comparación rápida.

INTERMEDIO

JAVA

String y texto

J079

RESPUESTA

""" ... """ (Java 15+). El compilador calcula la indentación MÍNIMA común (incluida la línea de cierre) y la quita.
Secuencias: \s preserva espacio al final de línea, \ evita el salto de línea (concatenar), \n explícito si se quiere.
No interpola variables: usar .formatted() o format.

BASICO

JAVA

String y texto

J081

¿Por qué concatenar String en un loop es un problema?

INTERMEDIO

JAVA

String y texto

J082

Regex en Java: Pattern y Matcher

INTERMEDIO

JAVA

String y texto

J083

¿Cuánta memoria ocupa un String y por qué importa?

AVANZADO

JAVA★

Colecciones

J084

ArrayList vs LinkedList

BASICO

JAVA★

Colecciones

J085

List vs Set vs Map

BASICO

JAVA

Colecciones

J086

Jerarquía del Collection Framework (mapa mental)

BASICO

JAVA

Colecciones

J087

HashSet: cómo decide si un elemento 'ya está'

BASICO

JAVA

Colecciones

J088

LinkedHashMap y LinkedHashSet: el punto medio perfecto

BASICO

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

JAVA

String y texto

J082

RESPUESTA

Pattern p = Pattern.compile("regex"); Matcher m = p.matcher(texto); m.find() itera coincidencias, m.group() la extrae, m.matches() exige coincidencia total.
 Grupos con paréntesis: m.group(1).
 Backslashes dobles en Java: "\\d+".
 Para validar simple: cadena.matches("regex").
 Compilar el Pattern UNA vez si se reutiliza (thread-safe).

INTERMEDIO

JAVA

String y texto

J081

RESPUESTA

Cada + crea un String nuevo y copia todo el contenido → $O(n^2)$ en el loop.
 Solución: StringBuilder en un hilo (append), StringBuffer solo si varios hilos comparten el builder (raro, está sincronizado).
 El compilador optimiza concatenaciones EN UNA MISMA expresión, no en loops.

INTERMEDIO

JAVA ★

Colecciones

J084

RESPUESTA

ArrayList: array dinámico; get por índice $O(1)$; insertar/borrar en medio $O(n)$ (desplaza elementos).
 LinkedList: doblemente enlazada; add/remove en extremos $O(1)$; get $O(n)$.
 En la práctica gana ArrayList casi siempre (localidad de caché).

BASICO

JAVA

String y texto

J083

RESPUESTA

Objeto String ≈ 24B cabecera + array de bytes. Desde Java 9 es byte[] con codificación compacta (Latin-1 si basta, UTF-16 si no) — menos memoria que el char[] antiguo.
 Duplicados en colecciones grandes → intern() o mapas de deduplicación.
 Substrings ya NO comparten array (copia) desde Java 7u6.

AVANZADO

JAVA

Colecciones

J086

RESPUESTA

Iterable → Collection → {List, Set, Queue}. Map aparte (no es Collection).
 List: ArrayList, LinkedList, Vector(legacy), CopyOnWriteArrayList.
 Set: HashSet, LinkedHashSet, TreeSet, EnumSet.
 Queue/Deque: ArrayDeque, PriorityQueue, LinkedList.
 Map: HashMap, LinkedHashMap, TreeMap, ConcurrentHashMap, Hashtable(legacy), EnumMap.

BASICO

JAVA ★

Colecciones

J085

RESPUESTA

List: ordenada, con índices, permite duplicados (ArrayList, LinkedList).
 Set: SIN duplicados (HashSet, TreeSet ordena por comparación).
 Map: pares clave→valor, claves únicas; no hereda de Collection (HashMap, TreeMap).

BASICO

JAVA

Colecciones

J088

RESPUESTA

Hash $O(1)$ + ORDEN DE INSERCIÓN predecible (lista doble enlazada interna).
 Access-order mode: new LinkedHashMap(16, 0.75f, true) reordena por acceso → base perfecta para una caché LRU (sobrescribir removeEldestEntry).

BASICO

JAVA

Colecciones

J087

RESPUESTA

1) calcula hashCode() del objeto para elegir bucket, 2) dentro del bucket compara con equals().
 Si sobrescribes equals sin hashCode (o el objeto MUTA su hash después de insertar) → contains() falla.
 Por eso: objetos clave inmutables o no mutar campos usados en hash tras insertar.

BASICO

JAVA

Colecciones

J089

ArrayDeque vs Stack vs LinkedList para pilas y colas

BASICO

JAVA

Colecciones

J090

PriorityQueue: comportamiento y trampas

BASICO

JAVA

Colecciones

J091

¿Cómo eliminar elementos de una lista mientras la recorres?

BASICO

JAVA★

Colecciones

J092

Comparable vs Comparator aplicado a ordenar colecciones

BASICO

JAVA

Colecciones

J093

List.of() vs Arrays.asList() vs new ArrayList<>()

BASICO

JAVA

Colecciones

J094

Colecciones no modificables vs inmutables

BASICO

JAVA★

Colecciones

J095

Métodos de Map modernos que quitan muchos ifs

BASICO

JAVA

Colecciones

J096

Iterator: contrato y uso correcto

BASICO

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

JAVA

Colecciones

J090

RESPUESTA

Heap binario: el elemento de MAYOR prioridad sale primero (el menor con orden natural).
 poll() $O(\log n)$, peek $O(1)$; NO está ordenada internamente (iterar no da orden).
 No admite null; con objetos necesita Comparable/Comparator.
 Uso: top-K, schedulers, Dijkstra.

BASICO

JAVA

Colecciones

J089

RESPUESTA

ArrayDeque: la opción recomendada — array circular, $O(1)$ amortizado en ambos extremos, menos GC.
 java.util.Stack hereda de Vector (sincronizado, legado, diseño malo) → evitar.
 LinkedList también implementa Deque pero con overhead de nodos y peor localidad de caché.
 Métodos: push/pop/peek, offer/poll/element, addFirst/last.

BASICO

JAVA ★

Colecciones

J092

RESPUESTA

lista.sort(comparator) o Collections.sort(lista).
 Comparator.comparing(User::getApellido).thenComparing(User::getEdad).reversed().
 nullsFirst/nullsLast para nulls.
 Comparar descendente: comparing(X).reversed() o .reversed() al final de la cadena completa (ojo: reversed() revierte TODO lo acumulado).

BASICO

JAVA

Colecciones

J091

RESPUESTA

1) Iterator: it.remove() — el clásico correcto.
 2) removeIf(predicado) — Java 8+, el más limpio (interno usa iterator).
 3) Recolectar en nueva lista (si no quieres mutar).
 MAL: for-each con lista.remove(x) → ConcurrentModificationException.

BASICO

JAVA

Colecciones

J094

RESPUESTA

Collections.unmodifiableList(vista): solo prohíbe modificar LA VISTA; si la lista de fondo cambia, la vista la refleja.
 List.of()/Set.of()/Map.of() (Java 9+): inmutables de verdad, copian los datos.
 Para exponer estado interno: copiar o devolver inmutable.

BASICO

JAVA

Colecciones

J093

RESPUESTA

List.of() (Java 9+): inmutable de verdad — add/remove/set lanzan UnsupportedOperationException; no permite nulls.
 Arrays.asList(): vista de tamaño FIJO sobre el array — set() sí funciona, add/remove no; refleja cambios del array.
 new ArrayList<>(): mutable completa.
 Pasar una de las fijas a código que muta → excepción en runtime.

BASICO

JAVA

Colecciones

J096

RESPUESTA

hasNext(), next(), remove() (único borrado seguro durante iteración).
 Solo se avanza hacia adelante, una pasada.
 ListIterator además: hasPrevious/previous, add, set, índices — bidireccional en List.
 Lanzan ConcurrentModificationException si la colección cambia por fuera.

BASICO

JAVA ★

Colecciones

J095

RESPUESTA

getOrDefault, putIfAbsent, computeIfAbsent (patrón memoización: map.computeIfAbsent(k, x -> new ArrayList<>()).add(v)), computeIfPresent, merge (contador: map.merge(k, 1, Integer::sum)), forEach(k,v), getOrDefault.
 put devuelve el valor ANTERIOR (null si no había).

BASICO

JAVA

Colecciones

J097

¿Por qué Map no extiende de Collection?

BASICO

JAVA

Colecciones

J098

entrySet vs keySet para recorrer un Map

BASICO

JAVA★

Colecciones

J099

¿Cómo funciona HashMap internamente?

INTERMEDIO

JAVA

Colecciones

J100

TreeSet/TreeMap: qué garantizan y qué cuestan

INTERMEDIO

JAVA

Colecciones

J101

ArrayList: cómo crece internamente

INTERMEDIO

JAVA

Colecciones

J102

fail-fast vs fail-safe en iteradores

INTERMEDIO

JAVA

Colecciones

J103

Complejidades que hay que memorizar

INTERMEDIO

JAVA★

Colecciones

J104

ConcurrentHashMap: qué lo hace diferente de un HashMap sincronizado

INTERMEDIO

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

JAVA

Colecciones

J098

RESPUESTA

Si necesitas clave Y valor: `entrySet()` — una sola búsqueda.
`keySet()` + `get(k)` hace DOBLE hash por entrada (innecesario).
`forEach((k,v) -> ...)` es el moderno.
`values()` cuando solo importan los valores.

BASICO

JAVA

Colecciones

J097

RESPUESTA

Porque es un mapeo clave→valor, no una colección de elementos: sus 'colecciones' derivadas son `keySet()`, `values()` y `entrySet()` (esas SÍ son `Collection/View`).
 Diseño: Map es otra abstracción distinta; se iteran sus entradas con `entrySet()`.

BASICO

JAVA

Colecciones

J100

RESPUESTA

Mantienen elementos ORDENADOS según `Comparable/Comparator` (árbol rojo-negro).
 Costo: `add/remove/contains` $O(\log n)$ vs $O(1)$ de `HashSet/HashMap`.
 Extras navegables: `first/last`, `floor/ceiling`, `headSet/tailSet`, `descendingSet`.
 No admiten null (con orden natural) y rompen el contrato si el comparador es inconsistente con equals.

INTERMEDIO

JAVA



Colecciones

J099

RESPUESTA

Array de buckets; índice = `hash(key) & (n-1)`.
 Colisiones: lista enlazada; desde Java 8 se balancea como árbol rojo-negro si el bucket llega a 8.
 Capacidad inicial 16, factor de carga 0.75 → `resize` duplica.
 Claves únicas; una clave null permitida.

INTERMEDIO

JAVA

Colecciones

J102

RESPUESTA

Fail-fast (`ArrayList`, `HashMap`): detectan modificación estructural durante la iteración (contador `modCount`) → `ConcurrentModificationException`. No es para hilos, es detección de bugs.
 Fail-safe: `CopyOnWriteArrayList` (itera sobre snapshot) y `ConcurrentHashMap` (iterador débilmente consistente) no lanzan.
 Eliminar mientras iteras: usar `iterator.remove()`, no `lista.remove()`.

INTERMEDIO

JAVA

Colecciones

J101

RESPUESTA

Array interno; al llenarse crece ~50% (nueva capacidad = `vieja*1.5`) y copia elementos.
`ensureCapacity()/constructor` con capacidad evita `reallocs` cuando sabes el tamaño.
`remove/contains` $O(n)$; `trimToSize()` libera sobrante.

INTERMEDIO

JAVA



Colecciones

J104

RESPUESTA

Segmentación/CAS: permite lecturas concurrentes sin lock y escrituras con locks finos por nodo — mucho mejor throughput.
 NO permite claves/valores null.
 Operaciones compuestas atómicas: `putIfAbsent`, `compute`, `merge`.
 Iterador débilmente consistente (no lanza CME).
`Collections.synchronizedMap` bloquea TODO el mapa → cuello de botella.

INTERMEDIO

JAVA

Colecciones

J103

RESPUESTA

`ArrayList`: `get` $O(1)$, `add` final amortizado $O(1)$, `add/remove` medio $O(n)$, `contains` $O(n)$.
`HashMap/HashSet`: $O(1)$ medio, $O(\log n)$ peor (treeify).
`TreeMap/TreeSet`: $O(\log n)$.
`LinkedList`: `add` extremos $O(1)$, `get i` $O(n)$.
`ArrayDeque`: extremos $O(1)$.
`PriorityQueue`: `offer/poll` $O(\log n)$.
`Binary search`: $O(\log n)$ (requiere orden).

INTERMEDIO

JAVA

Colecciones

J105

¿Cuándo CopyOnWriteArrayList?

INTERMEDIO

JAVA

Colecciones

J106

Colecciones y null: quién lo permite y quién no

INTERMEDIO

JAVA

Colecciones

J107

EnumSet y EnumMap: por qué los menciona todo entrevistador

INTERMEDIO

JAVA

Colecciones

J108

Collections utility: métodos que salvan vidas

INTERMEDIO

JAVA

Colecciones

J109

Capacidad inicial y factor de carga: ¿por qué afinarlos?

INTERMEDIO

JAVA

Colecciones

J110

¿List.contains con objetos propios devuelve false? Los 3 sospechosos

INTERMEDIO

JAVA

Colecciones

J111

remove(int index) vs remove(Object) en List: la trampa clásica

INTERMEDIO

JAVA

Colecciones

J112

¿Cómo funciona el resize de HashMap y su costo?

AVANZADO

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

JAVA

Colecciones

J106

RESPUESTA

Permiten null: ArrayList, LinkedList, HashMap (una clave null), HashSet, LinkedHashMap.
NO permiten: TreeMap/TreeSet (con orden natural, lanza NPE al comparar), ConcurrentHashMap, Hashtable, Queue concurrentes, List.of/Set.of/Map.of, PriorityQueue.
Diseño moderno: evitar nulls (Optional, valores default).

INTERMEDIO

JAVA

Colecciones

J105

RESPUESTA

Cuando hay MUCHÍSIMAS lecturas y casi cero escrituras (listeners, configs).
Cada escritura copia el array completo → costosa, pero los iteradores nunca ven CME ni cambios (snapshot).
Mal para listas que mutan seguido.

INTERMEDIO

JAVA

Colecciones

J108

RESPUESTA

sort, shuffle, reverse, swap, binarySearch (requiere orden), min/max con comparator, frequency, disjoint, unmodifiableX, synchronizedX, emptyList/emptyMap, singletonList, nCopies, addAll, rotate.
Muchos tienen equivalente moderno en la propia interfaz (list.sort, list.forEach).

INTERMEDIO

JAVA

Colecciones

J107

RESPUESTA

Implementaciones especiales para enums basadas en arrays de bits/array plano: muchísimo más rápidas y compactas que HashSet/HashMap de enums.
EnumSet.of(Dia.LUN, Dia.VIE), EnumMap<key, Valor>.
Orden = orden de declaración del enum.

INTERMEDIO

JAVA

Colecciones

J110

RESPUESTA

1) equals() no sobrescrito (compara referencias).
2) equals sobrescrito pero hashCode no (en HashSet/HashMap).
3) El objeto MUTÓ un campo que participa en equals/hashCode después de insertarlo.
Fix: equals+hashCode consistentes + objetos clave inmutables.

INTERMEDIO

JAVA

Colecciones

J109

RESPUESTA

HashMap default: capacidad 16, factor 0.75 → resize a las 12 inserciones (duplica y rehash).
Si sabes que guardarás 1M entradas: new HashMap<>(1_500_000) evita ~17 resizes.
Capacidad deseada = esperado / 0.75 (+1 para hash seguro).
ArrayList: igual idea con constructor.

INTERMEDIO

JAVA

Colecciones

J112

RESPUESTA

Al superar capacidad*loadFactor: crea array del doble y RE-DISTRIBUYE todas las entradas (rehash). Costo O(n) puntual, amortizado O(1) por inserción.
Java 8 conserva el orden relativo dentro de buckets (split por bit alto), antes invertía listas.
Con muchas inserciones previstas, presizar el mapa evita trabajo.

AVANZADO

JAVA

Colecciones

J111

RESPUESTA

List<Integer> nums: remove(1) elimina el ÍNDICE 1;
remove(Integer.valueOf(1)) elimina el OBJETO 1.
El compilador elige por el tipo estático del argumento.
Fuente infinita de bugs con listas de números.

INTERMEDIO

JAVA

Colecciones

J113

¿Qué son las vistas (views) de colecciones?

AVANZADO

JAVA

Genéricos

J114

¿Para qué existen los genéricos y qué problema arreglan?

BASICO

JAVA

Genéricos

J115

¿Por qué no hay List<int>?

BASICO

JAVA

Genéricos

J116

Métodos genéricos: sintaxis y cuándo

BASICO

JAVA

Genéricos

J117

Raw types y por qué el compilador los marca

BASICO

JAVA

Genéricos

J118

Herencia con genéricos: ¿List<Dog> es subtipo de List<Animal>?

BASICO

JAVA

Genéricos

J119

Genéricos, static y arrays: las tres prohibiciones

BASICO

JAVA

Genéricos

J120

Bounded types: <T extends Comparable<T>>

INTERMEDIO

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

JAVA

Genéricos

J114

RESPUESTA

Tipado seguro en compilación para colecciones/algoritmos reutilizables: `List<String>` garantiza que solo hay Strings (antes: raw types y casts a mansalva con `ClassCastException` en runtime). Eliminan casts explícitos y documentan el contrato del tipo.

BASICO

JAVA

Colecciones

J113

RESPUESTA

Objetos que ENVUELVEN otro sin copiar: `keySet()`, `subList()`, `unmodifiableList()`, `Arrays.asList()`. Cambios en el respaldo se reflejan en la vista (y a veces al revés: `subList` escribe en la original). Peligro: mutar el respaldo invalida la vista (`subList` → `ConcurrentModificationException`).

AVANZADO

JAVA

Genéricos

J116

RESPUESTA

`static <T> T primero(List<T> lista)` — el `<T>` va ANTES del tipo de retorno. Se infiere del contexto: `String s = primero(nombres)`. Útil cuando el método es genérico aunque la clase no, o para vincular tipos de argumento y retorno entre sí.

BASICO

JAVA

Genéricos

J115

RESPUESTA

Los genéricos solo aceptan tipos de referencia (type erasure + Object interno). `List<int>` no existe → `List<Integer>` con autoboxing (overhead de memoria/GC). Alternativas de rendimiento: `IntStream`, librerías (fastutil, Eclipse Collections) o arrays primitivos.

BASICO

JAVA

Genéricos

J118

RESPUESTA

NO (invarianza) — aunque `Dog` extends `Animal`. Si lo fuera, podrías meter un `Cat` en una `List<Dog>` vía la referencia `List<Animal>` y romper todo. Lo correcto: `List<? extends Animal>` para leer. `G = G<D>` solo si el tipo parametrizado coincide EXACTO: `ArrayList<Dog>` sí es `Collection<Dog>`.

BASICO

JAVA

Genéricos

J117

RESPUESTA

`List` sin parámetro = raw type: vuelve al mundo pre-Java 5 (acepta cualquier cosa, casts inseguros). NO es lo mismo que `List<Object>` (esta es tipada y segura). Usar raw types genera unchecked warnings y bugs silenciosos; solo aparecen por interoperabilidad legada.

BASICO

JAVA

Genéricos

J120

RESPUESTA

Limita `T` a un tipo o subtipos: `<T extends Number>` permite usar métodos de `Number` sobre `T`. Múltiples bounds: `<T extends Comparable<T> & Serializable>` (clase primero, interfaces después). En métodos: `static <T extends Comparable<T>> T max(List<T> lista)`.

INTERMEDIO

JAVA

Genéricos

J119

RESPUESTA

- 1) Campos static NO pueden usar `T` de la clase (`T` existe por instancia).
- 2) No se pueden crear arrays genéricos: `new T[10]`, `new List<String>[10]` (erasure no puede chequear; workaround: `(T[]) new Object[10]` con `@SuppressWarnings`).
- 3) No instanceof con tipos parametrizados (solo `List<?>` o clase cruda).

BASICO

JAVAGenéricosJ121

@SuppressWarnings("unchecked"): cuándo es legítimo

INTERMEDIO

JAVAGenéricosJ122

¿Qué es el type erasure y sus consecuencias?

AVANZADO

JAVAGenéricosJ123

Wildcard ? extends vs ? super (regla PECS)

AVANZADO

JAVAGenéricosJ124

¿Qué es el wildcard capture y el helper idiom?

AVANZADO

JAVASTarExcepcionesJ125

Checked vs unchecked exceptions

BASICO

JAVASTarExcepcionesJ126

¿Qué hace try-with-resources?

BASICO

JAVASTarExcepcionesJ127

Jerarquía completa de Throwable

BASICO

JAVAXExcepcionesJ128

throw vs throws vs Throwable

BASICO

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

JAVA

Genéricos

J122

RESPUESTA

El compilador BORRA los parámetros de tipo: `List<String>` y `List<Integer>` son la misma clase en runtime (`List`).
 Consecuencias: NO hay `new T()`, `T.class`, ni `instanceof List<String>`, ni arrays genéricos `new T[]`; los genéricos no participan en sobrecarga (`f(String)` vs `f(Integer)` ok, pero `f(List<String>)` vs `f(List<Integer>)` NO).
 Los tipos acotados se reemplazan por su bound y se insertan casts automáticos.

AVANZADO

JAVA

Genéricos

J121

RESPUESTA

Solo cuando TÚ garantizas la seguridad de tipo que el compilador no puede probar (casts de erasure, APIs legadas).
 Anotar la variable/método MÍNIMO posible, con comentario de por qué es seguro.
 Si se suprime a nivel de clase, se esconden bugs reales.

INTERMEDIO

JAVA

Genéricos

J124

RESPUESTA

El compilador trata cada uso de `?` como un tipo desconocido ÚNICO que no puede nombrarse: no puedes `lista.set(0, lista.get(0))` con `List<?>`.
 Truco: helper genérico privado `<T> void swapHelper(List<T> l){...}` y llamarlo desde el método con wildcard — el compilador captura el wildcard como `T` real.

AVANZADO

JAVA

Genéricos

J123

RESPUESTA

PECS: Producer Extends, Consumer Super.
`? extends T`: productor — LEES elementos como `T`, NO puedes agregar (solo null): `List<? extends Number>` acepta `List<Integer>`.
`? super T`: consumidor — ESCRIBES `T`, lo que lees es `Object`.
 Si solo lees: `extends`; si solo escribes: `super`; si ambos: tipo exacto.

AVANZADO

JAVA ★

Excepciones

J126

RESPUESTA

Cierra automáticamente cualquier recurso que implemente `AutoCloseable` al salir del bloque, incluso si hay excepción.
 Elimina el `finally` con `close()`.
 Si fallan el bloque y el `close()`, la del `close()` viaja como 'suppressed' adjunta a la principal.

BASICO

JAVA ★

Excepciones

J125

RESPUESTA

Checked (heredan de `Exception`): el compilador obliga a capturarlas o declararlas con `throws` — `IOException`, `SQLException`.
 Unchecked (heredan de `RuntimeException`): no obliga — `NullPointerException`, `IllegalArgumentException`.
 Error: fallos fatales de JVM (`OutOfMemoryError`).

BASICO

JAVA

Excepciones

J128

RESPUESTA

`throw`: sentencia que LANZA una instancia (`throw new IllegalArgumentException("...")`).
`throws`: en la firma, DECLARA que el método puede lanzar ciertas checked (puede listar unchecked pero es ruido).
`Throwable`: la superclase (solo se puede lanzar `Throwable` y subclases).

BASICO

JAVA ★

Excepciones

J127

RESPUESTA

`Throwable` → { `Error`, `Exception` }.
 Error: JVM irrecuperable (`OutOfMemoryError`, `StackOverflowError`) — no capturar.
`Exception` → `RuntimeException` (unchecked: `NPE`, `IllegalArgumentException`, `IndexOutOfBoundsException`) y las demás (checked: `IOException`, `SQLException`, `InterruptedException`).
 Custom: extender `Exception` (checked) o `RuntimeException` según semántica.

BASICO

JAVA

Excepciones

J129

¿Cuándo crear una excepción custom y con qué firma?

INTERMEDIO

JAVA

Excepciones

J130

Orden de catch y multi-catch

INTERMEDIO

JAVA

Excepciones

J131

¿Cuándo NO se ejecuta el finally?

INTERMEDIO

JAVA

Excepciones

J132

Anti-patrones clásicos de manejo de excepciones

INTERMEDIO

JAVA

Excepciones

J133

InterruptedException: qué significa y cómo tratarla

INTERMEDIO

JAVA★

Excepciones

J134

try-with-resources: reglas completas

INTERMEDIO

JAVA

Excepciones

J135

Rethrow y pérdida de stack trace

INTERMEDIO

JAVA

Excepciones

J136

assert: qué es y por qué no se usa en producción

INTERMEDIO

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

JAVA

Excepciones

J130

RESPUESTA

Los catch van de MÁS específico a MÁS general; catch de Exception antes que IOException no compila (bloqueo inalcanzable).

Multi-catch (Java 7+): catch (IOException | SQLException e) — e es implícitamente final; no pueden ser padre/hijo entre sí.

INTERMEDIO

JAVA

Excepciones

J129

RESPUESTA

Cuando el error tiene semántica de dominio (SaldoInsuficienteException).

Extender RuntimeException para no infectar firmas (regla moderna) o Exception si el llamador DEBE manejarla.

Constructor estándar: (String mensaje), (String mensaje, Throwable causa) — SIEMPRE preservar la causa para no perder el stack original.

INTERMEDIO

JAVA

Excepciones

J132

RESPUESTA

1) catch (Exception e) {} — tragarse el error (logging y rethrow o manejar).

2) catch(Exception e){ throw new RuntimeException(e.getMessage()) } — perder la causa (usar new RuntimeException(msg, e)).

3) Usar excepciones para control de flujo (costosas: llenan stack trace).

4) catch Throwable/Error.

5) Mensajes sin contexto ('error').

INTERMEDIO

JAVA

Excepciones

J131

RESPUESTA

Casi nunca: ni con return ni con excepción se salta.

Solo se pierde si: System.exit(), crash de la JVM, hilo daemon interrumpido al cerrar la JVM, o loop infinito dentro del try.

Trampa legada: un return dentro del finally SILENCIOSAMENTE descarta la excepción o el return del try — prohibido.

INTERMEDIO

JAVA ★

Excepciones

J134

RESPUESTA

Recursos: objetos AutoCloseable, separados por ';' (se cierran en orden INVERSO).

La variable es effectively final; se pueden declarar fuera y usar 'final var' (Java 9+).

Funciona también con try(recurso) sin catch.

Patrón: try (var conn = ...; var st = conn.createStatement()) {...}

INTERMEDIO

JAVA

Excepciones

J133

RESPUESTA

Se lanza en sleep/wait/join cuando el hilo es interrumpido (interrupt()).

Es CHECKED: fuerza a decidir.

Si no la manejas: restaurar la interrupción

(Thread.currentThread().interrupt()) y salir, para no 'tragarte' la señal de cancelación.

Nunca catch vacío.

INTERMEDIO

JAVA

Excepciones

J136

RESPUESTA

assert cond : "mensaje" — verificación de invariantes de desarrollo.

Se evalúa SOLO si la JVM corre con -ea (enableassertions); por defecto están DESACTIVADOS → en producción no protegen nada.

Para validación real:

IllegalArgumentException/Objects.requireNonNull (siempre activos).

INTERMEDIO

JAVA

Excepciones

J135

RESPUESTA

throw e; re-lanza la MISMA excepción conservando stack original.

throw new MiExcepcion(msg) sin causa: pierdes la historia → SIEMPRE wrap con causa: new MiExcepcion("contexto", e).

fillInStackTrace()/StackWalker para casos especiales.

INTERMEDIO

JAVA

Excepciones

J137

Objects utility para validar argumentos temprano

INTERMEDIO

JAVA

Excepciones

J138

Excepciones suprimidas (suppressed)

AVANZADO

JAVA

Excepciones

J139

¿Lanzar excepciones es caro? ¿Por qué?

AVANZADO

JAVA

Streams y lambdas

J140

Sintaxis de lambda: todas las variantes

BASICO

JAVA

Streams y lambdas

J141

Las 4 interfaces funcionales core y para qué sirven

BASICO

JAVA

Streams y lambdas

J142

Method references: las 4 formas

BASICO

JAVA

Streams y lambdas

J143

¿Qué es un Stream y en qué se diferencia de una colección?

BASICO

JAVA

Streams y lambdas

J144

Operaciones intermedias vs terminales y la pereza (lazy)

BASICO

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

JAVA

Excepciones

J138

RESPUESTA

En try-with-resources, si el try lanza X y el close() del recurso lanza Y: X se propaga y Y queda adjuntada.
 Recuperarlas: ex.getSuppressed() → Throwable[].
 No aplica al finally clásico (allí la excepción del finally REEMPLAZA a la original — se pierde).

AVANZADO

JAVA

Excepciones

J137

RESPUESTA

Objects.requireNonNull(x, "no puede ser null") → NPE con mensaje claro.
 Objects.requireNonNullElse(x, default).
 requireNonNull dentro de constructores/setters = fail-fast en el punto correcto (no un NPE críptico 10 líneas después).

INTERMEDIO

JAVA

Streams y lambdas

J140

RESPUESTA

(int x) -> x*1, (x) -> x*1, x -> x*1, () -> 42, (a, b) -> { int s = a+b; return s; }.
 Tipo inferido del contexto (interfaz funcional destino).
 Una expresión = return implícito; con {} se necesita return explícito.
 Variables capturadas: effectively final.

BASICO

JAVA

Excepciones

J139

RESPUESTA

Sí: se construye el stack trace (recorre la pila) en el constructor, aunque nadie lea el stack.
 En hot paths de validación frecuente (parsing masivo) pesa.
 Alternativa: resultados tipo Optional/Either o validación previa.
 Para errores REALES, el costo es irrelevante frente a la claridad.

AVANZADO

JAVA

Streams y lambdas

J142

RESPUESTA

1) Estático: Math::abs
 2) Instancia concreta: System.out::println
 3) Instancia del primer parámetro: String::toUpperCase (equivalente a s.toUpperCase())
 4) Constructor: ArrayList::new
 Pasa cualquier firma compatible con la interfaz funcional.

BASICO

JAVA

Streams y lambdas

J141

RESPUESTA

Function<T,R>: T→R (map). Predicate<T>: T→boolean (filter/test).
 Consumer<T>: T→void (efecto, forEach/accept). Supplier<T>: ()→T (produce, get).
 Variantes: BiFunction, BiPredicate, UnaryOperator (T→T), BinaryOperator (T,T→T), especializadas IntPredicate, ToIntFunction (evitan boxing).

BASICO

JAVA

Streams y lambdas

J144

RESPUESTA

Intermedias (devuelven Stream, perezosas): filter, map, flatMap, sorted, distinct, limit, skip, peek, takeWhile/dropWhile.
 Terminales (disparan el trabajo): collect, forEach, reduce, count, anyMatch/allMatch/noneMatch, findFirst/findAny, toArray, min/max.
 Sin terminal, NADA se ejecuta (fuente clásica de 'mi stream no hace nada').

BASICO

JAVA

Streams y lambdas

J143

RESPUESTA

Secuencia de elementos que NO almacena datos: procesa una fuente (colección, array, generador) de forma declarativa, perezosa y (opcionalmente) paralela.
 Colección = estructura de datos; Stream = pipeline de operaciones.
 Un stream se consume UNA vez; reutilizarlo lanza IllegalStateException.

BASICO

JAVA

Streams y lambdas

J145

map vs flatMap: la diferencia que siempre preguntan

BASICO

JAVA

Streams y lambdas

J146

reduce: las 3 formas y sus riesgos

BASICO

JAVA

Streams y lambdas

J147

Collectors que hay que dominar

BASICO

JAVA

Streams y lambdas

J148

groupBy: los niveles que salen en exámenes

BASICO

JAVA

Streams y lambdas

J149

Streams de primitivas: IntStream y amigos

BASICO

JAVA

Streams y lambdas

J150

Short-circuit: anyMatch, findFirst y streams infinitos

BASICO

JAVA

Streams y lambdas

J151

Optional en streams y su uso correcto

BASICO

JAVA

Streams y lambdas

J152

sorted(), distinct() y sus costos ocultos

BASICO

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

JAVA

Streams y lambdas

J146

RESPUESTA

1) Optional<T> reduce(accumBinaryOp)
 2) T reduce(identity, accum)
 3) <U> U reduce(identity, accum, combiner) (paralelo).
 Riesgos: identity NO neutro rompe paralelismo (usar 0 para suma, no -1); accum asociativo para paralelo.
 Para colecciones mejor Collectors.

BASICO

JAVA

Streams y lambdas

J145

RESPUESTA

map: 1 elemento → 1 elemento transformado (Stream<R>).
 flatMap: 1 elemento → MUCHOS elementos aplanados: lista de listas → lista plana (Stream<R> desde Streams internos).
 Ejemplo: pedidos.stream().flatMap(p -> p.getItems().stream()) = todos los ítems.

BASICO

JAVA

Streams y lambdas

J148

RESPUESTA

1 nivel: groupingBy(Persona::getCiudad) → Map<Ciudad, List<Persona>>.
 Con downstream: groupingBy(Persona::getCiudad, counting()) → Map<Ciudad, Long>.
 Con mapping: groupingBy(c, mapping(Persona::getNombre, toList())).
 Anidado: groupingBy(a, groupingBy(b, counting())) → conteo por dos dimensiones.
 partitioningBy es el caso booleano → Map<Boolean, List>.

BASICO

JAVA

Streams y lambdas

J147

RESPUESTA

toList/toSet/toCollection, joining(delimitador), counting, summingInt/averagingInt, summarizingInt (stats), groupingBy (y con downstream), partitioningBy (boolean), toMap(keyFn, valFn[, mergeFn, mapSupplier]), mapping, filtering, collectingAndThen, tteeing (dos recolectores + merge).
 toMap SIN mergeFn lanza IllegalStateException con claves duplicadas.

BASICO

JAVA

Streams y lambdas

J150

RESPUESTA

anyMatch/allMatch/noneMatch, findFirst/findAny, limit — cortan el procesamiento cuando ya tienen respuesta.
 Permite streams infinitos: Stream.iterate(1, i -> i*2).limit(10), Stream.generate(supplier).
 Java 9+: iterate con predicado de parada, Stream.ofNullable().forEach NO corta (usar takeWhile).

BASICO

JAVA

Streams y lambdas

J149

RESPUESTA

IntStream/LongStream/DoubleStream evitan boxing.
 Crear: IntStream.range(0,n) / rangeClosed, lista.stream().mapToInt(String::length).
 Extras: sum(), average(), summaryStatistics(), boxed() para volver a Stream<Integer>.
 map vs mapToInt: mapToObj, mapToDouble para cruzar.

BASICO

JAVA

Streams y lambdas

J152

RESPUESTA

sorted: estado con todo el stream en memoria + ordena O(n log n) — no es perezoso de verdad.
 distinct: mantiene un HashSet interno (necesita equals/hashCode).
 sorted después de un filter reduce datos; Comparator.comparing natural.
 En paralelo sorted es aún más caro.

BASICO

JAVA

Streams y lambdas

J151

RESPUESTA

findFirst/findAny/min/max/reduce devuelven Optional.
 Encadenar: stream.findFirst().map(String::toUpperCase).orElse("-").
 orElse vs orElseGet: orElse EVALÚA SIEMPRE el argumento (cuidado con llamadas costosas); orElseGet es perezoso.
 orElseThrow(supplier) para error explícito.

BASICO

JAVA

Streams y lambdas

J153

¿Cuándo NO usar streams?

BASICO

JAVA

Streams y lambdas

J154

Lambda vs clase anónima: diferencias finas

BASICO

JAVA

Streams y lambdas

J155

Comparador moderno con lambdas

BASICO

JAVA★

Streams y lambdas

J156

¿Qué son los Streams de Java 8?

INTERMEDIO

JAVA★

Streams y lambdas

J157

¿Para qué sirve Optional?

INTERMEDIO

JAVA

Streams y lambdas

J158

peek(), parallelStream() y sus trampas

AVANZADO

JAVA

Streams y lambdas

J159

Collectors.teeing y collectingAndThen (nivel pro)

AVANZADO

JAVA★

Concurrencia

J160

volatile vs synchronized

INTERMEDIO

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

JAVA

Streams y lambdas

J154

RESPUESTA

Lambda: solo interfaces funcionales, this se refiere a la clase ENVOLVENTE (no a la anónima), no crea una clase nueva por uso (invokeDynamic).

Anónima: sirve para clases/abstractas, tiene su propio this, puede tener estado/campos.

Para implementar interfaces funcionales simples, lambda siempre.

BASICO

JAVA

Streams y lambdas

J153

RESPUESTA

Lógica con múltiples efectos/branches complejos (mejor loop claro).

Mutación de estado externo dentro del pipeline (violación + bugs en paralelo).

Solo un par de operaciones triviales donde el loop es igual de claro.

Rendimiento crítico con elementos primitivos y acceso indexado (arrays + loops).

Legibilidad primero.

BASICO

JAVA ★

Streams y lambdas

J156

RESPUESTA

Pipeline funcional sobre colecciones:

fuente → intermedias PEREZOSAS (filter, map, sorted) →

TERMINAL (collect, forEach, reduce) que dispara la evaluación.

No mutan la fuente.

parallelStream() para procesar en paralelo (cuidado con orden y threads).

INTERMEDIO

JAVA

Streams y lambdas

J155

RESPUESTA

Comparator.comparing(P::getNombre) /
.comparingInt(P::getEdad), thenComparing, reversed(),
nullsFirst/nullsLast(Comparator.naturalOrder()).

En streams: sorted(Comparator.comparing(...)).

max/min de stream aceptan comparator directamente.

BASICO

JAVA

Streams y lambdas

J158

RESPUESTA

peek: pensado para DEBUG (ver elementos que pasan); NO garantiza ejecución salvo terminal, y abusar de él para mutar es frágil.

parallelStream: usa ForkJoinPool.commonPool — SOLO para fuentes fácilmente dividibles y operaciones SIN estado, asociativas, sin efectos secundarios; NO para I/O; puede degradar (compartido en toda la JVM).

findAny/forEach en paralelo no preservan orden; usar
forEachOrdered.

AVANZADO

JAVA ★

Streams y lambdas

J157

RESPUESTA

Contenedor que expresa 'puede no haber valor' en el tipo de retorno; evita null y NPE.

Métodos: map, filter, orElse / orElseGet, orElseThrow, ifPresent.

Antipatrón: Optional como campo o parámetro.

Nunca .get() sin comprobar.

INTERMEDIO

JAVA ★

Concurrencia

J160

RESPUESTA

volatile: garantiza VISIBILIDAD (las escrituras de un hilo se ven al instante) pero NO atomicidad — i++ sigue roto.

synchronized: exclusión mutua + visibilidad mediante lock de monitor; bloquea un método o bloque.

INTERMEDIO

JAVA

Streams y lambdas

J159

RESPUESTA

teeing(f1, f2, merge): combina dos recolecciones sobre el mismo stream — ej. máx y mín juntos, o porcentaje: teeing(counting(), summingInt(...), (n, s) -> 100.0*s/n).

collectingAndThen(collector, finisher): transforma el resultado final — ej. toList → List.copyOf, o toSet → TreeSet::new.

AVANZADO

JAVA

★

Concurrencia

J161

¿Qué es ExecutorService?

INTERMEDIO

JAVA

★

Concurrencia

J162

¿Qué añade CompletableFuture sobre Future?

INTERMEDIO

JAVA

Concurrencia

J163

Ciclo de vida de un hilo en Java

INTERMEDIO

JAVA

Concurrencia

J164

Race condition: qué es y el ejemplo de siempre

INTERMEDIO

JAVA

Concurrencia

J165

synchronized: método vs bloque y sobre QUÉ se bloquea

INTERMEDIO

JAVA

Concurrencia

J166

ReentrantLock vs synchronized: cuándo el extra vale la pena

INTERMEDIO

JAVA

Concurrencia

J167

AtomicInteger y CAS: cómo funcionan sin locks

INTERMEDIO

JAVA

Concurrencia

J168

CountDownLatch vs CyclicBarrier vs Semaphore

INTERMEDIO

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

JAVA ★
RESPUESTA

Concurrencia

J162

Composición NO bloqueante:

- thenApply / thenCompose / thenCombine encadenan etapas
- callbacks sin .get() bloqueante
- errores con exceptionally / handle
- combina tareas paralelas

Es la base de la programación asíncrona en Java moderno.

INTERMEDIO

JAVA ★
RESPUESTA

Concurrencia

J161

Pool de hilos que ejecuta tareas sin crear hilos a mano.

Envías Callable/Runnable y recibes Future con el resultado.

Fábricas: newFixedThreadPool, newCachedThreadPool, newScheduledThreadPool.

Siempre cerrarlo: shutdown() / shutdownNow().

INTERMEDIO

JAVA
RESPUESTA

Concurrencia

J164

El resultado depende del entrelazado de hilos. Clásico:

contador++ no es atómico (leer-modificar-escribir: 3 pasos) → dos hilos pueden leer el mismo valor y perder una suma.

Fixes: synchronized, AtomicInteger (CAS), LongAdder para contadores de alta contención.

No confundir con deadlock.

INTERMEDIO

JAVA
RESPUESTA

Concurrencia

J163

NEW → (start) RUNNABLE → (corriendo / listo) → BLOCKED (esperando lock) → WAITING (wait/join sin timeout) →

TIMED_WAITING (sleep/wait con timeout) → TERMINATED.

No se puede reiniciar un hilo terminado

(IllegalThreadStateException): usa run() en lugar de start() NO crea hilo.

INTERMEDIO

JAVA
RESPUESTA

Concurrencia

J166

ReentrantLock: tryLock(timeout) (no bloquear indefinidamente), lockInterruptibly(), fairness (cola FIFO), múltiples Condition.

Costo: debes hacer unlock() en finally SIEMPRE.

Regla: synchronized por defecto; ReentrantLock solo para esas necesidades avanzadas.

INTERMEDIO

JAVA
RESPUESTA

Concurrencia

J165

Método de instancia: lock del objeto (this). Método static: lock del objeto Class (Clase.class).

Bloque: synchronized(otroObjeto) { } — lock específico, mínimo alcance recomendado.

Reentrante: el mismo hilo puede tomar el lock de nuevo.

Todo lo compartido entre hilos debe protegerse con el MISMO lock.

INTERMEDIO

JAVA
RESPUESTA

Concurrencia

J168

CountDownLatch: cuenta regresiva de UNA VEZ (await espera a que count llegue a 0) — 'espera a que N tareas terminen'; no reutilizable.

CyclicBarrier: N hilos se ESPERAN MUTUAMENTE en un punto; reutilizable; acción barrier opcional.

Semaphore(N): permisos — limita acceso concurrente a un recurso (pool de conexiones); acquire/release.

INTERMEDIO

JAVA
RESPUESTA

Concurrencia

J167

Compare-And-Swap: operación atómica de hardware — 'si el valor actual es X, ponme Y' — reintentando en loop si falló.

AtomicInteger.incrementAndGet() es lock-free y muy rápido.

Limitación: operaciones compuestas complejas (compareAndSet encadenado, acumulación con lógica → synchronized o LongAdder).

Otros: AtomicReference, AtomicBoolean, AtomicLong.

INTERMEDIO

JAVA★

Concurrencia

J169

BlockingQueue: el patrón productor-consumidor en 3 líneas

INTERMEDIO

JAVA

Concurrencia

J170

ScheduledExecutorService: tareas periódicas correctas

INTERMEDIO

JAVA

Concurrencia

J171

¿Qué es la contención (contention) y cómo se reduce?

INTERMEDIO

JAVA

Concurrencia

J172

@Async en Spring vs CompletableFuture manual

INTERMEDIO

JAVA

Concurrencia

J173

Crear hilos: Runnable vs Thread vs Callable

AVANZADO

JAVA★

Concurrencia

J174

wait/notify/notifyAll: contrato exacto

AVANZADO

JAVA

Concurrencia

J175

Deadlock: las 4 condiciones y cómo prevenirlo

AVANZADO

JAVA

Concurrencia

J176

LongAdder vs AtomicLong

AVANZADO

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

JAVA

Concurrencia

J170

RESPUESTA

`scheduleAtFixedRate(tarea, ini, periodo)`: intenta ejecutar cada 'periodo' (fijo desde el inicio; si la tarea tarda más, se solapan las intenciones).

`scheduleWithFixedDelay(tarea, ini, delay)`: espera 'delay' ENTRE fin e inicio (nunca solapa).

Si la tarea LANZA excepción, las siguientes se CANCELAN → `try/catch` interno obligatorio.

INTERMEDIO

JAVA



Concurrencia

J169

RESPUESTA

`ArrayBlockingQueue<Integer> cola = new`

`ArrayBlockingQueue<>(100);`

Productor: `cola.put(item)` — bloquea si llena. Consumidor:

`cola.take()` — bloquea si vacía.

Desacopla velocidades, acota memoria (bounded), sin wait/notify manual.

Variantes: `LinkedBlockingQueue`, `PriorityBlockingQueue`, `SynchronousQueue` (entrega directa).

INTERMEDIO

JAVA

Concurrencia

J172

RESPUESTA

`@Async`: Spring envuelve el método con un proxy y lo ejecuta en un executor configurado (`TaskExecutor`) — necesidad:

`@EnableAsync` y NO self-invocation (mismo límite que `@Transactional`).

`CompletableFuture`: control fino de composición y executor por llamada.

Ambos devuelven `Future/CompletableFuture` si se quiere el resultado. Nunca anotar métodos `private`.

INTERMEDIO

JAVA

Concurrencia

J171

RESPUESTA

Tiempo que los hilos gastan esperando locks/recursos compartidos.

Reducir: scopes más cortos de `synchronized`, locks finos/por-striping (`ConcurrentHashMap`), estructuras lock-free (CAS), datos por hilo (`ThreadLocal`), inmutabilidad, `LongAdder`, particionar el estado.

Medir: profilers de contención, JFR.

INTERMEDIO

JAVA



Concurrencia

J174

RESPUESTA

Se llaman DENTRO de `synchronized` sobre el MISMO objeto monitor, sino `IllegalMonitorStateException`.

`wait()`: libera el lock y espera; `notify()`: despierta UNO; `notifyAll()`: despierta todos (el más seguro — evita hilos dormidos para siempre).

Siempre en un `while(!condicion) wait()` — para despertares espurios.

Hoy se prefiere `java.util.concurrent` (`BlockingQueue`, `CountDownLatch`).

AVANZADO

JAVA

Concurrencia

J173

RESPUESTA

`Runnable`: tarea sin resultado, `new Thread(runnable).start()`.

`Thread`: heredar (acopla tarea con ejecución — evitar).

`Callable<V>`: devuelve resultado y lanza excepciones checked — se usa con `ExecutorService/Future`.

Recomendado: nunca crear hilos 'a mano'; usar pools (executors).

AVANZADO

JAVA

Concurrencia

J176

RESPUESTA

`LongAdder`: mantiene celdas múltiples que se suman al leer (`sum()`) — muchísimo mejor bajo ALTA CONTENCIÓN de escritura; lecturas ligeramente más caras.

`AtomicLong`: todos los hilos CAS sobre un solo valor → colisiones.

Contadores de métricas → `LongAdder`.

AVANZADO

JAVA

Concurrencia

J175

RESPUESTA

Exclusión mutua, retención y espera, no expulsión, espera circular.

Prevención: adquirir locks en ORDEN GLOBAL consistente, usar `tryLock` con timeout (`ReentrantLock`), evitar locks anidados, granularidad fina.

Detección: dump de hilos (jstack) muestra 'Found one Java-level deadlock'.

AVANZADO

JAVA

Concurrencia

J177

ExecutorService: ciclo de vida correcto

AVANZADO

JAVA★

Concurrencia

J178

¿Cómo dimensionar un pool de hilos?

AVANZADO

JAVA

Concurrencia

J179

Future: qué ofrece y qué le falta

AVANZADO

JAVA

Concurrencia

J180

CompletableFuture: recetario de métodos

AVANZADO

JAVA

Concurrencia

J181

ThreadLocal: qué es y su fuga de memoria clásica

AVANZADO

JAVA

Concurrencia

J182

interrupt(): qué hace y qué no

AVANZADO

JAVA

Concurrencia

J183

Hilos daemon vs usuario

AVANZADO

JAVA

Concurrencia

J184

Colecciones thread-safe: mapa rápido de decisión

AVANZADO

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

JAVA ★
RESPUESTA

Concurrencia

J178

CPU-bound: núcleos + 1
(`Runtime.getRuntime().availableProcessors()`).
I/O-bound: más hilos porque pasan bloqueados: núcleos * (1 + tiempoEspera/tiempoCPU) o empíricamente decenas.
En servicios web: cada pool por dependencia (bulkhead) con colas ACOTADAS (`LinkedBlockingQueue` sin límite = OOM).
Medir y ajustar.

AVANZADO

JAVA
RESPUESTA

Concurrencia

J177

Crear: `Executors.newFixedThreadPool(n)`, `newCachedThreadPool` (cuidado: hilos ilimitados), `newSingleThreadExecutor`, `newScheduledThreadPool`.
Ejecutar: `submit` (Callable → Future), `execute` (Runnable).
Cerrar: `shutdown()` (termina tareas pendientes, no acepta nuevas) → `awaitTermination` → `shutdownNow()` (interrumpe) en finally.
No cerrar = hilos no-daemon impiden salir de la JVM.

AVANZADO

JAVA
RESPUESTA

Concurrencia

J180

`supplyAsync(tarea, executor) / runAsync`.
Transformar: `thenApply` (sync), `thenApplyAsync`.
Encadenar: `thenCompose` (evita CF<CF<T>>), `thenCombine` (une dos independientes).
Consumir: `thenAccept/thenRun`.
Errores: `exceptionally`, `handle`, `whenComplete`.
Todos los *Async aceptan executor propio (¡sino commonPool!).

AVANZADO

JAVA
RESPUESTA

Concurrencia

J179

`get()` (bloquea, con timeout opcional),
`cancel(mayInterruptIfRunning)`, `isDone/isCancelled`.
Falta: callbacks no bloqueantes, composición, manejo de errores elegante → por eso `CompletableFuture`.
Trampa: `get()` sin timeout puede colgar para siempre.

AVANZADO

JAVA
RESPUESTA

Concurrencia

J182

NO detiene el hilo: pone un flag y hace que `sleep/wait/join/bloqueos` interrumpibles lancen `InterruptedException`.
El hilo decide cooperar: revisar `Thread.currentThread().isInterrupted()` en loops largos.
Limpiar el flag al capturar la excepción si vas a continuar:
`Thread.currentThread().interrupt()`.

AVANZADO

JAVA
RESPUESTA

Concurrencia

J181

Variable con valor POR HILO (cada hilo ve su copia): `user context`, transacciones, `SimpleDateFormat` legado.
Uso: `ThreadLocal.withInitial(() -> new ...)`.
Fuga: en pools de hilos los hilos viven para siempre → si no haces `remove()` en el finally, el valor queda anclado para siempre.
Siempre `try { } finally { threadLocal.remove(); }`.

AVANZADO

JAVA
RESPUESTA

Concurrencia

J184

Map concurrente → `ConcurrentHashMap` (o `ConcurrentSkipListMap` si ordenado).
Lista concurrente → `CopyOnWriteArrayList` (muchas lecturas) o listas sincronizadas pequeñas.
Cola productor/consumidor → `ArrayBlockingQueue` (acotada) / `LinkedBlockingQueue` / `ConcurrentLinkedQueue` (lock-free).
Set concurrente → `ConcurrentHashMap.newKeySet()`.
NUNCA `Vector/Hashtable` para código nuevo.

AVANZADO

JAVA
RESPUESTA

Concurrencia

J183

Daemon: hilo de servicio (GC, monitores) — la JVM sale SIN esperarlos; `setDaemon(true)` ANTES de `start()`.
Usuario: la JVM espera a que terminen.
Trampa: tareas críticas en daemon pueden cortarse a la mitad al cerrar la app.

AVANZADO

JAVA

Concurrencia

J185

¿Por qué 'double-checked locking' era difícil y cómo se arregla?

AVANZADO

JAVA

Concurrencia

J186

Happens-before: la idea central del Java Memory Model

AVANZADO

JAVA

Concurrencia

J187

¿Por qué volatile NO hace atómico i++?

AVANZADO

JAVA

Concurrencia

J188

Singleton thread-safe: las 4 formas canónicas

AVANZADO

JAVA

Concurrencia

J189

ForkJoinPool y work-stealing: para qué existe

AVANZADO

JAVA

Concurrencia

J190

Virtual threads (Java 21): qué cambian de verdad

AVANZADO

JAVA

Concurrencia

J191

¿Qué es el 'pinning' de virtual threads?

AVANZADO

JAVA

Concurrencia

J192

Inmutabilidad como estrategia de concurrencia

AVANZADO

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

JAVA

Concurrencia

J186

RESPUESTA

Garantía de VISIBILIDAD: si la acción A happens-before B, todo lo que A escribió es visible para B.

Construyen happens-before: release/acquire del mismo lock (unlock→lock), escritura/lectura volatile, start()/join() de un hilo, métodos de concurrencia del paquete j.u.c.

Sin una de estas relaciones, los cambios pueden no verse NUNCA (reordenamiento + cachés de CPU).

AVANZADO

JAVA

Concurrencia

J185

RESPUESTA

Singleton lazy: sin volatile, un hilo puede ver un objeto a medio construir (reordenamiento de instrucciones).

Fix: campo private static volatile Instancia instancia.

Alternativas mejores: holder idiom (clase interna estática — lazy y thread-safe por classloading) o enum singleton.

AVANZADO

JAVA

Concurrencia

J188

RESPUESTA

1) Enum singleton (SimpleSingleton.INSTANCE) — la más segura.

2) Holder idiom: clase interna estática con la instancia (lazy por classloading, sin locks).

3) Double-checked locking con volatile.

4) Campo static final inicializado al cargar (eager).

En Spring, el contenedor ya maneja singletons — casi no se escribe a mano.

AVANZADO

JAVA

Concurrencia

J187

RESPUESTA

volatile garantiza visibilidad y orden, no atomicidad compuesta: i++ sigue siendo leer+sumar+escribir (3 pasos que pueden entrelazarse).

Fix: AtomicInteger.incrementAndGet(), o synchronized.

volatile perfecto para: flags de estado (running = false), publicación de referencias inmutables.

AVANZADO

JAVA

Concurrencia

J190

RESPUESTA

Hilos ligeros gestionados por la JVM (no 1:1 con hilos del SO): millones posibles.

Para I/O-bound: Thread.ofVirtual().start() o Executors.newVirtualThreadPerTaskExecutor(); el hilo se desmonta al bloquearse en I/O → concurrencia masiva con código SÍNCRONO simple (no callbacks).

NO acelera CPU-bound. Evitar pooling de virtual threads (uno por tarea) y sincronización prolongada (pinning).

AVANZADO

JAVA

Concurrencia

J189

RESPUESTA

Pool que DIVIDE tareas recursivamente (fork/join) y los hilos ROBAN trabajo de las colas de otros (work-stealing) → buen balanceo para divide & conquer (procesar árboles, parallel streams lo usan).

RecursiveTask<V>/RecursiveAction; invoke()/compute().

Es el commonPool por defecto de los parallel streams.

AVANZADO

JAVA

Concurrencia

J192

RESPUESTA

Objetos inmutables NO necesitan locks: compartirlos es gratis y seguro (publicación segura vía volatile/final).

Diseño: state en records inmutables, cambios = nueva instancia (copy-on-write mentalidad).

Es la base de ConcurrentHashMap, CopyOnWrite*, akka/actores y del estilo funcional.

AVANZADO

JAVA

Concurrencia

J191

RESPUESTA

Un virtual thread montado sobre un carrier no puede desmontarse si ejecuta synchronized prolongado o native (JNI) bloqueante → bloquea su hilo del SO (¡el recurso escaso que querías ahorrar!).

Java 21+ y 24 han mitigado mucho el caso synchronized.

Workaround histórico: ReentrantLock en lugar de synchronized en hot paths de I/O.

AVANZADO

JAVA★

JVM y rendimiento

J193

¿Diferencia entre JDK, JRE y JVM?

BASICO

JAVA

JVM y rendimiento

J194

¿Qué hace la JVM al ejecutar tu .class?

BASICO

JAVA★

JVM y rendimiento

J195

Estructura de memoria de la JVM

BASICO

JAVA★

JVM y rendimiento

J196

¿Cómo funciona el Garbage Collector?

INTERMEDIO

JAVA★

JVM y rendimiento

J197

Stack vs Heap

INTERMEDIO

JAVA

JVM y rendimiento

J198

G1 GC: cómo funciona en 5 puntos

AVANZADO

JAVA

JVM y rendimiento

J199

ZGC / Shenandoah vs G1: ¿cuándo low-latency?

AVANZADO

JAVA★

JVM y rendimiento

J200

OutOfMemoryError: los tipos y qué significan

AVANZADO

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

JAVA

JVM y rendimiento

J194

RESPUESTA

- 1) ClassLoader carga el bytecode (loading, linking —verify/prepare/resolve—, initializing).
 - 2) El intérprete ejecuta bytecode línea a línea.
 - 3) El compilador JIT (C1 cliente, C2 optimizador) detecta hotspots y compila a código máquina nativo con optimizaciones (inlining, escape analysis, loop unrolling).
 - 4) El GC gestiona el heap.
- Resultado: arranque lento, luego muy rápido.

BASICO

JAVA



JVM y rendimiento

J193

RESPUESTA

JVM: ejecuta el bytecode (.class) y da independencia de plataforma.
 JRE = JVM + librerías para EJECUTAR programas.
 JDK = JRE + herramientas para DESARROLLAR (javac, javadoc, jdb).
 Desarrollas con JDK; ejecutas con JRE.

BASICO

JAVA



JVM y rendimiento

J196

RESPUESTA

Rastrea objetos alcanzables desde las raíces (hilos, static); lo no alcanzable se elimina del heap.
 Heap generacional: Young (Eden + Survivor, GCs menores) y Old (GC mayor).
 System.gc() solo es una sugerencia: no garantiza nada.

INTERMEDIO

JAVA



JVM y rendimiento

J195

RESPUESTA

Heap: objetos — Young (Eden + 2 Survivor) y Old/Tenured.
 Metaspace (Java 8+): metadatos de clases (reemplaza PermGen), en memoria nativa.
 Stacks: uno por hilo (frames, variables locales).
 Code cache: JIT nativo.
 Además: direct buffers, thread stacks.
 Flags: -Xms/-Xmx (heap), -XX:MaxMetaspaceSize, -Xss (stack).

BASICO

JAVA

JVM y rendimiento

J198

RESPUESTA

- GC por defecto desde Java 9.
- 1) Divide el heap en regiones (~2048) iguales.
 - 2) Marca concurrente para saber vivos.
 - 3) Jóvenes: pausas cortas (copy entre regiones).
 - 4) Mixtas: recoge regiones viejas con más basura primero.
 - 5) Objetivo de pausa configurable: -XX:MaxGCPauseMillis=200.
- Compacta (sin fragmentación como CMS).

AVANZADO

JAVA



JVM y rendimiento

J197

RESPUESTA

Stack: por hilo, guarda frames de métodos y variables locales/primitivas (LIFO, muy rápido).
 Heap: compartido, guarda objetos y arrays; lo gestiona el GC.
 OutOfMemoryError → heap. StackOverflowError → recursión sin fin.

INTERMEDIO

JAVA



JVM y rendimiento

J200

RESPUESTA

Java heap space: heap lleno de objetos vivos (fuga o -Xmx chico).
 GC overhead limit exceeded: >98% de tiempo en GC recuperando <2%.
 Metaspace: demasiadas clases (generación dinámica, fuga de classloaders).
 Unable to create native thread: límite de hilos del SO.
 Direct buffer memory: NIO buffers fuera del heap.
 Diagnóstico: heap dump
 (-XX:+HeapDumpOnOutOfMemoryError) + Eclipse MAT.

AVANZADO

JAVA

RESPUESTA

JVM y rendimiento

J199

ZGC/Shenandoah: pausas < 1-10 ms INDEPENDIENTES del tamaño del heap (TB de heap), casi todo concurrente.
 Costo: algo más de CPU total (concurrente) y footprint.
 Para: APIs con SLO de latencia estricta, heaps enormes.
 G1: balance por defecto excelente.
 Activar: -XX:+UseZGC.

AVANZADO

JAVA

JVM y rendimiento

J201

Memory leak en Java (¡con GC!): las fuentes típicas

AVANZADO

JAVA

JVM y rendimiento

J202

Referencias fuertes, suaves, débiles y fantasmas

AVANZADO

JAVA

JVM y rendimiento

J203

Classloading: delegación padre-hijo

AVANZADO

JAVA

JVM y rendimiento

J204

¿Qué mide un profiler y cuáles hay en el kit JVM?

AVANZADO

JAVA

JVM y rendimiento

J205

Flame graphs: cómo leerlos en 30 segundos

AVANZADO

JAVA

JVM y rendimiento

J206

Autoboxing y rendimiento: el caso típico

AVANZADO

JAVA

JVM y rendimiento

J207

¿Qué es JMH y por qué System.nanoTime() no basta para benchmarks?

AVANZADO

JAVA

JVM y rendimiento

J208

Escape analysis y optimizaciones del JIT

AVANZADO

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

JAVA

JVM y rendimiento

J202

RESPUESTA

Strong: nunca se recolecta mientras sea alcanzable.
SoftReference: se limpia ANTES de OOM — cachés de memoria sensibles.
WeakReference: se limpia en el próximo GC si solo hay referencias débiles — WeakHashMap (metadatos asociados a objetos).
PhantomReference: notificación post-mortem (cleanup de nativo; reemplazado por Cleaner).

AVANZADO

JAVA

JVM y rendimiento

J201

RESPUESTA

Colecciones estáticas que solo crecen (cachés sin eviction), ThreadLocal sin remove en pools, listeners/handlers no desregistrados, classloaders con despliegues calientes (tomcat), objetos anclados por campos estáticos, subcadenas/arrays grandes retenidos.
Síntoma: heap sube escalonadamente y nunca baja; Full GCs frecuentes.
Herramienta: heap dump + dominators tree (MAT).

AVANZADO

JAVA

JVM y rendimiento

J204

RESPUESTA

CPU por método (hotspots), memoria/retención, hilos, I/O.
Herramientas: JFR (Java Flight Recorder) + JMC — bajo overhead, producción; jconsole/visualvm (local); async-profiler (flame graphs).
Metodología: medir ANTES de optimizar; regla 90/10 (pocos métodos concentran el tiempo).

AVANZADO

JAVA

JVM y rendimiento

J203

RESPUESTA

Jerarquía: Bootstrap → Platform → Application (→ custom).
Delegación: el loader pide AL PADRE primero; si no lo encuentra, carga él — evita duplicar clases core y asegura que java.lang.String sea 'el' String.
Carga perezosa: una clase se inicializa al PRIMER uso activo (new, acceso static).
Aplicaciones web contenedores usan loaders por app (aislamiento).

AVANZADO

JAVA

JVM y rendimiento

J206

RESPUESTA

En loops intensivos: List<Long> suma via boxed (cada operación crea objetos) vs long[] o IntStream.range().sum() (primitivos).
long total = 0; for (long x : array) total += x — gana por goleada.
Medir con JMH si es crítico; microoptimizar sin datos es adivinar.

AVANZADO

JAVA

JVM y rendimiento

J205

RESPUESTA

Cada caja = un método en ejecución; ancho = proporción de tiempo mostrado.
X axis: stack apilado (abajo = llamador); el ANCHO total es lo que importa.
Platachas anchas planas = hotspot claro.
En modo diff (antes/después) las mejoras se ven por colores.

AVANZADO

JAVA

JVM y rendimiento

J208

RESPUESTA

El JIT analiza si un objeto ESCAPA del método: si no, puede eliminar la asignación (scalar replacement), hacer inlining de métodos calientes y unrolling de loops.
Inlining es la reina: habilita todas las demás — métodos GRANDES o final/mega-mórficos (2 tipos reales máx) se inlinean mejor.
Consecuencia: abstracciones 'gratis' si JIT las elimina; megamorfismo mata el inlining.

AVANZADO

JAVA

JVM y rendimiento

J207

RESPUESTA

JMH (OpenJDK Microbenchmark Harness): maneja warmup de JIT, dead-code elimination, aislamiento de forks, distribución estadística.
nanoTime a mano mide el código optimizado-away o sin calentar (primeras ejecuciones lentas).
Uso: anotaciones @Benchmark, @State, fork, warmup — resultados en ops/time con percentiles.

AVANZADO

JAVA

JVM y rendimiento

J209

Flags JVM que todo backend debería conocer

AVANZADO

JAVA

JVM y rendimiento

J210

String deduplication y compact strings

AVANZADO

JAVA

JVM y rendimiento

J211

Cachés de CPU y localidad de datos: por qué ArrayList le gana a LinkedList

AVANZADO

JAVA

★

Java moderno

J212

¿Qué es un record (Java 14+)?

BASICO

JAVA

Java moderno

J213

switch de expresión (Java 14+): qué aporta

BASICO

JAVA

Java moderno

J214

HttpClient (Java 11+): uso básico

BASICO

JAVA

Java moderno

J215

Nuevos métodos de colección (Java 8-11) que debes usar

BASICO

JAVA

Java moderno

J216

Unnamed variables y patterns (..) — Java 21/22

BASICO

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

JAVA

JVM y rendimiento

J210

RESPUESTA

Compact strings (Java 9+): byte[] Latin-1 cuando basta → ~50% menos memoria en textos occidentales.
 String deduplication (G1, -XX:+UseStringDeduplication): GC comparte arrays de chars idénticos entre Strings (para apps con millones de Strings duplicados).
 Complementarias a intern().

AVANZADO

JAVA

JVM y rendimiento

J209

RESPUESTA

-Xms/-Xmx (heap inicial/máx, iguales en prod),
 -XX:MaxMetaspaceSize, -XX:+HeapDumpOnOutOfMemoryError
 -XX:HeapDumpPath, -XX:MaxGCPauseMillis (G1), -Xlog:gc* (logs GC), -XX:+UseStringDeduplication (G1), -agentlib:jwp (debug),
 -Dsystem.properties.
 Prod: habilitar JFR:
 -XX:StartFlightRecording=duration=60s,filename=...

AVANZADO

JAVA



Java moderno

J212

RESPUESTA

Clase inmutable de datos en una línea:
 record Point(int x, int y) {}
 Genera constructor, accesores (x(), y()), equals, hashCode y toString.
 No puede extender otras clases. Ideal para DTOs y valores.

BASICO

JAVA

JVM y rendimiento

J211

RESPUESTA

La CPU trae líneas de caché (64B) contiguas: recorrer un array es prefetch-friendly.
 LinkedList: cada nodo es un objeto suelto en el heap → cache misses en cada salto.
 Regla práctica: ArrayList casi siempre gana incluso en inserciones medias por la localidad.
 Datos contiguos = rendimiento (primitivos > objetos).

AVANZADO

JAVA

Java moderno

J214

RESPUESTA

HttpClient client = HttpClient.newHttpClient();
 HttpRequest req = HttpRequest.newBuilder(URI.create(url)).header("Accept", "application/json").GET().build();
 HttpResponse<String> resp = client.send(req, BodyHandlers.ofString()); — también sendAsync(CompletableFuture).
 Soporta HTTP/2, timeouts (connectTimeout, request timeout), y reemplaza HttpURLConnection.

BASICO

JAVA

Java moderno

J213

RESPUESTA

Devuelve VALOR: int dias = switch(mes) { case ENE, MAR -> 31; default -> 30; }.
 Flecha -> sin fall-through; caso múltiple con comas; yield para bloques: case X -> { ... yield resultado; }.
 Exhaustividad: con enums/sealed, cubrir todos los casos reemplaza el default.
 Menos bugs de break olvidado.

BASICO

JAVA

Java moderno

J216

RESPUESTA

Usa '_' cuando no necesitas la variable: catch (Exception _), for (var _ : lista), pattern Punto(int _, int y).
 Expressa intención: 'esto aquí no importa' y evita nombres throw-away (ignored, unused).
 Además unnamed classes y el main compacto (Java 21+ preview → 25 final) para scripts.

BASICO

JAVA

Java moderno

J215

RESPUESTA

forEach/forEachRemaining, removeIf, replaceAll, computeIfAbsent/compute/merge (Map), getOrDefault, putIfAbsent, List.sort, Iterator.removeRemaining? — y en Java 21: SequencedCollection (getFirst/getLast/reversed()) para List/Deque).
 List.of/Set.of/Map.of + ofEntries y Map.entry (Java 9).

BASICO

JAVA

Java moderno

J217

Anotaciones: @Retention, @Target y cómo se leen en runtime

BASICO

JAVA

Java moderno

J218

Reflection: usos legítimos y costos

BASICO

JAVA

Java moderno

J219

JShell y scripts Java: qué permiten hoy

BASICO

JAVA

Java moderno

J220

¿Qué cambió con las versiones LTS y el ritmo semestral?

BASICO

JAVA

Java moderno

J221

Variedades de Optional (Java 9-11): or, ifPresentOrElse, stream

INTERMEDIO

JAVA

Java moderno

J222

Pattern matching para switch (Java 21): ejemplo con sealed

AVANZADO

JAVA

Java moderno

J223

Record patterns (Java 21): deconstrucción

AVANZADO

JAVA

Java moderno

J224

¿Qué es JPMS (Project Jigsaw, Java 9) y cuándo importa?

AVANZADO

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

JAVA

RESPUESTA

Java moderno

J218

Instanciar/invocar en runtime por nombre: `Class.forName()`, `getDeclaredConstructor().newInstance()`, `method.invoke(obj, args)` — `setAccessible(true)` para privados.
Usos: frameworks (DI, ORMs, serializers), testing de privados (mejor no).
Costos: lento vs llamada directa, rompe encapsulación, choca con modules/native-image. Cachear Method/Constructor si se usa mucho.

BASICO

JAVA

RESPUESTA

Java moderno

J217

@Retention: SOURCE (solo compilador), CLASS (default, no runtime), RUNTIME (reflexión las ve — Spring/JPA usan RUNTIME).
@Target: dónde aplica (METHOD, FIELD, TYPE, PARAMETER).
Leerlas: `clase.getMethod("x").getAnnotation(MiAnon.class)`.
Es el mecanismo que usan los frameworks para configurar sin XML.

BASICO

JAVA

RESPUESTA

Java moderno

J220

Ritmo: feature release cada 6 meses; LTS cada 2 años: 8 (2014), 11 (2018), 17 (2021), 21 (2023), 25 (2025).
21 trae: virtual threads (final), pattern matching switch, record patterns, sequenced collections.
Empresas: saltan de LTS a LTS; features 'preview' se estabilizan en 1-3 releases.

BASICO

JAVA

RESPUESTA

Java moderno

J219

JShell (Java 9+): REPL — probar APIs, calcular, explorar sin proyecto.
Scripts: `java script.java` ejecuta un .java directo (single-file source), con clases locales/unnamed y main compacto para utilidades rápidas.
En contenedores: `jshell -c` o imagen con JDK (no JRE slim).

BASICO

JAVA

RESPUESTA

Java moderno

J222

sealed interface Forma permits Circulo, Cuadrado.
`String nombre = switch(forma) { case Circulo c -> "círculo r=" + c.radio(); case Cuadrado c when c.lado() > 10 -> "cuadrado grande"; default -> "otro"; }`.
Guardas con when; el compilador verifica exhaustividad — agregar un subtipo rompe la compilación de todos los switch, que es BUENO.

AVANZADO

JAVA

RESPUESTA

Java moderno

J221

or(supplier): Optional alternativo perezoso.
ifPresentOrElse(consumer, runnable): hace una cosa si hay valor, otra si no (adiós if/else con isPresent).
stream(): convierte Optional<Stream> flat en un stream — ideal en flatMap: `repositorio.findById(id).stream()`...
isEmpty() complementa isPresent.

INTERMEDIO

JAVA

RESPUESTA

Java moderno

J224

Sistema de módulos: module-info.java declara requires y exports — encapsulación REAL (paquetes no exportados no son accesibles ni por reflexión por defecto).
Importa para: JDK modularizado (jlink para runtimes mínimos), bibliotecas fuertemente encapsuladas.
La mayoría de apps de negocio NO usan módulos propios, pero conviven con el JDK modular.

AVANZADO

JAVA

RESPUESTA

Java moderno

J223

`record Punto(int x, int y) { }`
`if (obj instanceof Punto(int x, int y)) { usar x, y }` — extrae componentes directamente.
Anidados: `case Linea(Punto(int x1,_), Punto p2)` — '_' es unnamed variable (Java 21+).
Elimina getters intermedios en lógica de descomposición.

AVANZADO

JAVA

Java moderno

J225

Proxy dinámico: cómo funciona Spring AOP por dentro

AVANZADO

JAVA

Java moderno

J226

Virtual threads + Structured Concurrency: el nuevo modelo

AVANZADO

JAVA

I/O y serialización

J227

java.io vs java.nio: la diferencia en una frase

BASICO

JAVA

I/O y serialización

J228

Files y Path: el recetario de operaciones de archivo

BASICO

JAVA

I/O y serialización

J229

Encoding: por qué especificar SIEMPRE el charset

BASICO

JAVA

I/O y serialización

J230

Lectura de archivos grandes sin reventar memoria

BASICO

JAVA

I/O y serialización

J231

transient y serialVersionUID

BASICO

JAVA

I/O y serialización

J232

Scanner vs BufferedReader para leer entrada

BASICO

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

JAVA

Java moderno

J226

RESPUESTA

StructuredTaskScope (preview): lanza subtareas virtuales y garantiza que NINGUNA quede huérfana — scope.join() espera; shutdownOnFailure o shutdownOnSuccess para cancelar hermanos.
Código concurrente con la claridad del secuencial: try (var scope = new StructuredTaskScope.ShutdownOnFailure()) { ... scope.join().throwIfFailed(); }

AVANZADO

JAVA

Java moderno

J225

RESPUESTA

Proxy.newProxyInstance(loader, interfaces, handler) crea en runtime una clase que implementa las interfaces y delega al InvocationHandler.
Spring: si tu bean implementa interfaces usa proxy JDK (por interfaz); si no, CGLIB (subclase generada).
Por eso: métodos final/static/private no se interceptan y self-invocation no pasa por el proxy.

AVANZADO

JAVA

I/O y serialización

J228

RESPUESTA

Path p = Path.of("dir", "archivo.txt"); Files.readAllLines(p), readString(p), write(p, bytes), exists, createDirectories, copy/move (REPLACE_EXISTING), delete, walk (stream recursivo), list, size, getLastModifiedTime.
Buffered: Files.newBufferedReader/Writer (charset explícito!).

BASICO

JAVA

I/O y serialización

J227

RESPUESTA

I/O clásico: streams bloqueantes orientados a byte/char.
NIO: channels + buffers + selectores (I/O no bloqueante) y un FileSystem moderno (Path/Files).
Para apps comunes: Files/Paths (NIO.2) para operaciones de archivo, streams para contenido.

BASICO

JAVA

I/O y serialización

J230

RESPUESTA

NUNCA readAllLines/readAllBytes para GBs.
Usar: Files.lines(p) (Stream perezoso, cerrar con try-with-resources), BufferedReader.lines(), o streaming por líneas.
Escribir con BufferedWriter y flush por lotes.
Para binario: InputStream.transferTo(out) copia en buffer.

BASICO

JAVA

I/O y serialización

J229

RESPUESTA

DefaultCharset depende del SO/locale (Windows-1252, UTF-8...) → acentos rotos entre entornos ('Ä©').
SIEMPRE: new InputStreamReader(in, StandardCharsets.UTF_8), Files.readString(p, UTF_8), getBytes(UTF_8).
Java 18+: UTF-8 es el default de la plataforma (JEP 400), pero explícito sigue siendo mejor.

BASICO

JAVA

I/O y serialización

J232

RESPUESTA

Scanner: parseo cómodo (nextInt, nextLine) pero lento y con trampa nextInt + nextLine (deja el \n — consumirlo).
BufferedReader.readLine: rápido, devuelve String, parseas con Integer.parseInt.
En challenges/entrada masiva: BufferedReader + StringTokenizer (split manual rápido).

BASICO

JAVA

I/O y serialización

J231

RESPUESTA

transient: campo NO se serializa (contraseñas, conexiones, cachés derivadas) — al deserializar queda null/0.
serialVersionUID: huella de versión de la clase; si cambia entre serializar y deserializar → InvalidClassException. Fijarlo explícito para controlar compatibilidad.
writeObject/readObject custom para control fino.

BASICO

JAVA

I/O y serialización

J233

Temp files, cleaning y Files.walk

BASICO

JAVA

I/O y serialización

J234

Serializable: cómo funciona y sus peligros

INTERMEDIO

JAVA

Fechas y hora

J235

java.time: las clases que hay que conocer (y qué reemplazan)

BASICO

JAVA

Fechas y hora

J236

¿Todas inmutables y thread-safe? ¿Y por qué importa?

BASICO

JAVA

Fechas y hora

J237

Duration vs Period vs Instant

BASICO

JAVA

Fechas y hora

J238

DateTimeFormatter: parseo y formato seguro

BASICO

JAVA

Fechas y hora

J239

Zonas horarias: la regla de oro al guardar fechas

BASICO

JAVA

Fechas y hora

J240

Operaciones frecuentes con LocalDate

BASICO

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

JAVA

I/O y serialización

J234

RESPUESTA

implements Serializable → ObjectOutputStream serializa el GRAFO completo de objetos (transitivamente).

Peligros: deserialización de datos no confiables = vulnerabilidad grave (gadget chains); versionado frágil (serialVersionUID); expone representación interna.

Alternativas modernas: JSON (Jackson), Protobuf/Avro.

Si lo usas: serialVersionUID fijo + transient para no serializar campos.

INTERMEDIO

JAVA

I/O y serialización

J233

RESPUESTA

Files.createTempFile(prefix, suffix) / createTempDirectory — en %TEMP%.

Borrar: Files.delete en finally, o deleteOnExit() (solo crece la lista; no para largos).

Files.walk(ruta) → Stream<Path> recursivo (cerrarlo); filter por extensiones; Files.walkFileTree para control fino (visitor).

BASICO

JAVA

Fechas y hora

J236

RESPUESTA

Sí: java.time es inmutable — cada operación devuelve NUEVA instancia (fecha.plusDays(1) NO muta).

Comparte instancias entre hilos sin riesgo (Date/Calendar no).

Trampa clásica: fecha = fecha.plusDays(1); sin reasignar se 'pierde' el cambio.

BASICO

JAVA

Fechas y hora

J235

RESPUESTA

LocalDate (solo fecha), LocalTime, LocalDateTime, ZonedDateTime (con zona), OffsetDateTime (con offset), Instant (punto en la línea UTC, timestamps), Duration (tiempo/horas), Period (días/meses/años), DateTimeFormatter, ZonedId.

Reemplazan a Date/Calendar (mutables, meses 0-based, zona confusa — legado).

BASICO

JAVA

Fechas y hora

J238

RESPUESTA

DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm") o constantes ISO (ISO_LOCAL_DATE).

Formato: fecha.format(fmt). Parseo: LocalDate.parse("2024-01-31") (ISO) o LocalDate.parse(s, fmt).

IMMUTABLE y thread-safe (SimpleDateFormat NO lo era — bug clásico).

Locales: ofPattern(..., Locale.forLanguageTag("es")).

BASICO

JAVA

Fechas y hora

J237

RESPUESTA

Duration: cantidad de TIEMPO en escala de reloj (horas, minutos, segundos, nanos) — Duration.between(ini, fin).

Period: cantidad en CALENDARIO (años, meses, días) — Period.between(f1, f2).

Instant: punto exacto en UTC (epoch seconds+nanos) — para timestamps y machine time.

ZonedDateTime para mostrar a humanos con zona.

BASICO

JAVA

Fechas y hora

J240

RESPUESTA

plusDays/plusMonths/plusYears/plus(Period), minusX, withDayOfMonth, atStartOfDay(), atTime(h,m), toLocalDate/toLocalDateTime, isBefore/isAfter/isEqual, until(otro, ChronoUnit.DAYS).

Primer/último del mes: with(TemporalAdjusters.firstDayOfMonth()), lastDayOfMonth.

lengthOfMonth(), isLeapYear().

BASICO

JAVA

Fechas y hora

J239

RESPUESTA

Guardar y transmitir en UTC (Instant) o timestamps; convertir a zona SOLO al mostrar.

ZonedDateTime.now(ZonedId.of("America/Lima")); conversión: zdt.withZoneSameInstant(otraZona).

Nunca hacer aritmética de fechas con Strings ni asumir la zona del servidor.

OffsetDateTime vs ZonedDateTime: offset fijo vs reglas de zona (DST).

BASICO

JAVA

Fechas y hora

J241

**LocalDate vs LocalDateTime vs Instant:
¿cuál guardar para un evento?**

BASICO

JAVA

Fechas y hora

J242

**¿Cómo migrar código legado con Date y
Calendar?**

BASICO

JAVA

Fechas y hora

J243

Comparar y medir intervalos con java.time

BASICO

JAVA

Fechas y hora

J244

**Reloj inyectable (Clock) para código
testable**

BASICO

JAVA ★

Snippets: ¿qué imprime?

J245

**Integer a=127,b=127; Integer c=128,d=128;
a==b ? c==d ?**

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J246

**String s="hola"; String t="ho"+"la"; s==t ?
new String("hola")==s ?**

INTERMEDIO

JAVA ★

Snippets: ¿qué imprime?

J247

**List<Integer> l = new
ArrayList<>(List.of(1,2,3)); l.remove(1);
¿qué queda?**

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J248

**try { return "try"; } finally {
System.out.println("fin"); } — ¿orden de
salida?**

INTERMEDIO

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

JAVA

Fechas y hora

J242

RESPUESTA

Convertir: `date.toInstant()` (Java 8+), `Instant→Date`:
`Date.from(instant)`.
Calendar: `cal.toInstant()`.
Nuevas APIs no aceptan `Date`: convertir en la frontera del sistema legado y usar `java.time` por dentro.
Evitar mezclar en el mismo dominio.

BASICO

JAVA

Fechas y hora

J241

RESPUESTA

`Instant` (o timestamp UTC en DB): eventos de sistema, auditoría, `createdAt` — sin ambigüedad.
`LocalDate`: fechas de calendario puras (cumpleaños, vencimientos sin hora).
`LocalDateTime`: hora local sin zona (horarios de tienda) — solo si TODOS comparten zona.
Regla: si cruza zonas o sistemas → `Instant/ZonedDateTime`.

BASICO

JAVA

Fechas y hora

J244

RESPUESTA

`Clock.systemDefaultZone()` en producción; `Clock.fixed(instant, zone)` en tests.
Instanciar servicios con `Clock` y usar `Instant.now(clock)` / `LocalDate.now(clock)` → pruebas deterministas de 'hoy', vencimientos y expiraciones sin sleeps.

BASICO

JAVA

Fechas y hora

J243

RESPUESTA

`isBefore/isAfter/isEqual` entre objetos del mismo tipo.
`Duration.between(t1, t2).toMillis()` / `toMinutes()`.
`ChronoUnit.DAYS.between(f1, f2)`, `MONTHS.between`.
between exige tipos compatibles (`LocalDate` con `LocalDate`, `Instant` con `Instant`).

BASICO

JAVA

Snippets: ¿qué imprime?

J246

RESPUESTA

`true` y `false`.
"ho"+"la" es constante plegada en COMPILACIÓN → misma referencia del pool que `s`.
`new String("hola")` crea un objeto nuevo en el heap → `== false` (equals sería `true`).
`s.intern() == new String("hola").intern()` → `true`.

INTERMEDIO

JAVA



Snippets: ¿qué imprime?

J245

RESPUESTA

`true` y `false`.
`Integer.valueOf` cachea [-128, 127] → `a` y `b` son el MISMO objeto.
128 crea objetos distintos → `==` compara referencias y da `false`.
Lección: wrappers SIEMPRE `equals()`.

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J248

RESPUESTA

Imprime 'fin' y LUEGO devuelve "try": el `finally` corre ANTES de que el `return` del `try` complete la salida del método.
Si el `finally` hiciera `return "finally"`, ese valor GANA y se silencian excepciones pendientes (mala práctica).

INTERMEDIO

JAVA



Snippets: ¿qué imprime?

J247

RESPUESTA

[1, 3] — `remove(1)` elimina el ÍNDICE 1 (el 2), no el valor 1.
Para eliminar el valor: `remove(Integer.valueOf(1))`.
El overload elegido depende del tipo ESTÁTICO del argumento:
`int` → índice; `Integer` → objeto.

INTERMEDIO

JAVA ★

Snippets: ¿qué imprime?

J249

```
System.out.println(0.1+0.2 == 0.3); y  
System.out.println(1/2);
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J250

```
Integer x = null; int y = (condicion) ? x :  
0; — ¿qué pasa si condicion es true?
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J251

```
char c = 'A'; System.out.println(c + 1);  
System.out.println("" + c + 1);
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J252

```
Map<Integer,String> m = new  
HashMap<>(); m.put(1,"a");  
m.put(null,"b"); ¿lanza algo? ¿y en  
TreeMap?
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J253

```
static int x=10; static { x=20; } ¿en qué  
orden se ejecuta con new Clase()?
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J254

```
double d = Double.NaN; d == d — ¿true o  
false?
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J255

```
String s = "a"; s.concat("b");  
System.out.println(s);
```

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J256

```
List.of("a").add("b") — ¿qué pasa?
```

INTERMEDIO

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

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J250

NullPointerException.

El ternario con un brazo int y otro Integer fuerza UNBOXING de ambos → desempaqueta el null.

Fix: int y = (cond && x != null) ? x : 0; o evitar mezclar primitivo y wrapper en ternarios.

INTERMEDIO

JAVA



RESPUESTA

Snippets: ¿qué imprime?

J249

false y 0.

0.1+0.2 = 0.30000000000000004 en binario IEEE754 → comparar doubles con epsilon o BigDecimal.

1/2 es división ENTERA (trunca a 0); 1.0/2 sería 0.5.

INTERMEDIO

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J252

HashMap: perfecto, admite UNA clave null (y valores null) — no lanza nada.

TreeMap.put(null,...) → NullPointerException (comparar null con orden natural rompe el árbol).

ConcurrentHashMap → NullPointerException también.

INTERMEDIO

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J251

66 y A1.

char + int = aritmética (promoción a int → 65+1).

String + char inicia concatenación → "A1" (todo lo que sigue se concatena).

INTERMEDIO

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J254

false.

NaN es lo único distinto de sí mismo en IEEE754.

Detectarlo: Double.isNaN(d). Y NaN en un HashSet puede duplicarse (equals falsificado por comparación de bits:

Double.valueOf(NaN).equals(NaN) sí es true — inconsistencia deliberada).

INTERMEDIO

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J253

En orden de aparición del archivo: primero x=10 (inicializador de campo), luego el bloque static x=20.

Todo el bloque estático corre UNA vez al cargar la clase, ANTES de cualquier instancia y antes de main.

Reordenar líneas cambia el resultado final → pregunta clásica de examen.

INTERMEDIO

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J256

UnsupportedOperationException EN RUNTIME.

List.of crea colecciones INMUTABLES; cualquier mutación lanza excepción.

Si necesitas mutable: new ArrayList<>(List.of("a")) — constructor copia.

INTERMEDIO

JAVA

RESPUESTA

Snippets: ¿qué imprime?

J255

"a" — String es INMUTABLE: concat devuelve un NUEVO String que se descarta al no asignarlo.

Lo mismo con toUpperCase, replace, trim...

Solo si haces s = s.concat("b") cambia la variable (a un objeto nuevo).

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J257

HashSet<Point> set: agrego p (x=1), luego p.x=2, set.contains(p nuevo con x=2) → ?

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J258

System.out.println(10 + 20 + "" + 10 + 20);

INTERMEDIO

JAVA★

JDBC y localización

J259

¿Qué es JDBC?

BASICO

JAVA

JDBC y localización

J260

Localización (i18n): Locale, ResourceBundle y NumberFormat

BASICO

JAVA

JDBC y localización

J261

JDBC: flujo básico de una consulta (código)

INTERMEDIO

JAVA★

JDBC y localización

J262

PreparedStatement vs Statement: ¿por qué SIEMPRE PreparedStatement?

INTERMEDIO

JAVA

JDBC y localización

J263

Transacciones en JDBC plano

INTERMEDIO

SPRING BOOT★

Núcleo IoC

S001

¿Qué son IoC y la inyección de dependencias?

BASICO

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

JAVA

Snippets: ¿qué imprime?

J258

RESPUESTA

`"302010"`.De izquierda a derecha: $10+20$ (aritmética) = 30; al tocar el String "" todo lo demás se CONCATENA: $30 + "10" + "20" \rightarrow 302010$.

Evaluó de expresión: la aritmética solo ocurre mientras no haya String en la cadena.

INTERMEDIO

JAVA

Snippets: ¿qué imprime?

J257

RESPUESTA

false (o comportamiento inconsistente): el objeto `MUTÓ` su hashCode DESPUÉS de insertarse — quedó en un bucket que ya no le corresponde.

El set puede no encontrar ni eliminar el elemento (remove falla silenciosamente).

Lección: campos que participan en hash>equals deben ser inmutables.

INTERMEDIO

JAVA

JDBC y localización

J260

RESPUESTA

Locale: identifica idioma+país (`new Locale("es", "PE")`, `Locale.getDefault()`).ResourceBundle: archivos `messages_es.properties`, `messages_en.properties` → `bundle.getString("saludo")` según el Locale.

NumberFormat/DateFormat: formatean números, monedas y fechas según el locale (1.234,56 vs 1,234.56).

OCP lo pregunta poco, pero aparece en apps multi-idioma.

BASICO

JAVA



JDBC y localización

J259

RESPUESTA

Java Database Connectivity: la API estándar de Java para hablar con bases de datos relacionales.

Estructura: `DriverManager/DataSource` → `Connection` → `Statement` → `ResultSet`.

Cada motor (Postgres, MySQL...) provee su DRIVER (jar) que implementa la API.

Es la base sobre la que trabajan JPA/Hibernate y JdbcTemplate.

BASICO

JAVA



JDBC y localización

J262

RESPUESTA

`PreparedStatement`: la consulta se PRE-COMPIILA y los parámetros van aparte (?) → immune a SQL Injection y más rápida si se reutiliza.Statement con concatenación: `'WHERE nombre = ' + input + ''` → `input='; DROP TABLE users; --` = desastre.

Regla de seguridad: jamás concatenar input del usuario en SQL.

INTERMEDIO

JAVA

JDBC y localización

J261

RESPUESTA

```
try (Connection cn = DriverManager.getConnection(url, user, pass);
    PreparedStatement ps = cn.prepareStatement("SELECT nombre FROM users WHERE id = ?")) {
    ps.setLong(1, id);
    try (ResultSet rs = ps.executeQuery()) {
        while (rs.next()) System.out.println(rs.getString("nombre"));
    }
}
```

Recursos `AutoCloseable` → `try-with-resources` obligatorio.

INTERMEDIO

SPRING BOOT



Núcleo IoC

S001

RESPUESTA

IoC: el contenedor (`ApplicationContext`) crea, cablea y gestiona el ciclo de vida de los beans; tú no haces new.

DI: las dependencias llegan inyectadas (constructor > setter > campo).

Beneficios: bajo acoplamiento, testeo con mocks, cambiar implementaciones fácil.

BASICO

JAVA

JDBC y localización

J263

RESPUESTA

Por defecto está en `autocommit` (cada sentencia es su tx).

Control manual:

```
cn.setAutoCommit(false);
try { ...updates...; cn.commit(); }
catch (Exception e) { cn.rollback(); }
```

Es lo que `@Transactional` hace por ti debajo (con `DataSourceTransactionManager`).

INTERMEDIO

SPRING BOOT

★

Núcleo IoC

S002

@Component vs @Service vs @Repository vs @Controller

BASICO

SPRING BOOT

★

Núcleo IoC

S003

¿Por qué inyectar por constructor y no por campo?

BASICO

SPRING BOOT

Núcleo IoC

S004

@Bean vs @Component

BASICO

SPRING BOOT

Núcleo IoC

S005

¿Qué son las interfaces @Enable* ?

BASICO

SPRING BOOT

Núcleo IoC

S006

Component scan: paquetes base y filtros

BASICO

SPRING BOOT

Núcleo IoC

S007

Bean con nombre: convenciones y referencia por nombre

BASICO

SPRING BOOT

★

Núcleo IoC

S008

Scopes de beans en Spring

INTERMEDIO

SPRING BOOT

★

Núcleo IoC

S009

Ciclo de vida de un bean

INTERMEDIO

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

SPRING BOOT



Núcleo IoC

S003

RESPUESTA

Constructor:

- dependencias obligatorias explícitas
- campos pueden ser final
- testeable sin Spring (new + mocks)
- detecta dependencias circulares al arrancar.

@Autowired en campo: oculta dependencias y exige reflejo en tests. Spring recomienda constructor.

BASICO

SPRING BOOT



Núcleo IoC

S002

RESPUESTA

Son estereotipos: todas registran un bean.

@Component: genérico.

@Service: lógica de negocio (semántica).

@Repository: traduce excepciones de persistencia a DataAccessException.

@Controller: capa web. @RestController = @Controller +

@ResponseBody (JSON).

BASICO

SPRING BOOT

RESPUESTA

Núcleo IoC

S005

Interruptores que importan configuración: @EnableAsync,

@EnableScheduling, @EnableCaching,

@EnableTransactionManagement, @EnableFeignClients...

Internamente usan @Import para registrar beans/proxies necesarios.

En Spring Boot muchas vienen activadas por autoconfiguración.

BASICO

SPRING BOOT

RESPUESTA

Núcleo IoC

S004

@Component: sobre TU clase; se registra por escaneo de componentes.

@Bean: método dentro de @Configuration que DEVUELVE la instancia; para clases de terceros que no puedes anotar.

@Configuration con proxy de métodos reutiliza llamadas a @Bean (respetando singleton).

BASICO

SPRING BOOT

RESPUESTA

Núcleo IoC

S007

Nombre default: primera letra minúscula del nombre de clase (userService) o del método @Bean.

Custom: @Component("miNombre") / @Bean("miNombre").

Injectar por nombre: @Qualifier o campo cuyo nombre coincide — cuando tipo y nombre coinciden no hace falta qualifier.

BASICO

SPRING BOOT

RESPUESTA

Núcleo IoC

S006

@SpringBootApplication escanea desde SU paquete hacia abajo — clase main mal ubicada = beans no encontrados (error #1 de principiantes).

Personalizar: @ComponentScan(basePackages, includeFilters, excludeFilters).

Beans fuera del paquete base: registrarlos con @Import o @Bean.

BASICO

SPRING BOOT



Núcleo IoC

S009

RESPUESTA

Instanciación → inyección de dependencias → callbacks Aware →

BeanPostProcessor (before) → @PostConstruct →

afterPropertiesSet → LISTO → al cerrar: @PreDestroy.

Los BeanPostProcessors envuelven los beans: así funcionan

@Transactional, @Async, @Lazy.

INTERMEDIO

SPRING BOOT



Núcleo IoC

S008

RESPUESTA

singleton (default): una instancia por contenedor.

prototype: nueva por cada petición/inyección.

request, session, websocket: por contexto web.

Trampa: inyectar prototype dentro de un singleton se resuelve

UNA vez → usar ObjectProvider o @Lookup.

INTERMEDIO

SPRING BOOT

Núcleo IoC

S010

BeanFactory vs ApplicationContext

INTERMEDIO

SPRING BOOT

Núcleo IoC

S011

@Primary y @Qualifier: resoluciones de ambigüedad

INTERMEDIO

SPRING BOOT

Núcleo IoC

S012

@Lazy: qué hace y sus dos usos clave

INTERMEDIO

SPRING BOOT

Núcleo IoC

S013

@DependsOn, @Order y @PostConstruct/@PreDestroy

INTERMEDIO

SPRING BOOT

Núcleo IoC

S014

¿Cómo resuelve Spring las dependencias circulares?

AVANZADO

SPRING BOOT

Núcleo IoC

S015

BeanPostProcessor vs BeanFactoryPostProcessor

AVANZADO

SPRING BOOT

Núcleo IoC

S016

FactoryBean: qué es y cuándo aparece

AVANZADO

SPRING BOOT

Núcleo IoC

S017

@Lookup y ObjectProvider: inyección de prototype en singleton

AVANZADO

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

SPRING BOOT

Núcleo IoC

S011

RESPUESTA

Cuando 2+ beans implementan la misma interfaz, la inyección por tipo es ambigua → `NoUniqueBeanDefinitionException`.
`@Primary`: el 'por defecto'.
`@Qualifier("nombre")`: elige explícito en el punto de inyección.
 Alternativa: inyectar `Map<String, MiServicio>` (todos por nombre) o `List<MiServicio>` (todos en orden `@Order`).

INTERMEDIO

SPRING BOOT

Núcleo IoC

S010

RESPUESTA

`BeanFactory`: contenedor básico, crea beans perezosamente (al pedirlos).
`ApplicationContext`: la superinterfaz que usan las apps — añade eventos, internacionalización, recursos, y beans singleton se crean AL ARRANCAR (eager).
 En Spring Boot obtienes un `ApplicationContext` automáticamente.

INTERMEDIO

SPRING BOOT

Núcleo IoC

S013

RESPUESTA

`@DependsOn("beanA")`: fuerza orden de creación sin dependencia de tipo (side effects, caches de arranque).
`@Order/@Ordered`: orden relativo en colecciones inyectadas, interceptores y filtros.
`@PostConstruct`: init tras inyección; `@PreDestroy`: cleanup al cerrar contexto (solo singletons; prototype no gestiona destrucción).

INTERMEDIO

SPRING BOOT

Núcleo IoC

S012

RESPUESTA

1) Diferir la creación del bean hasta su primer uso (bean caro o poco usado).
 2) ROMPER dependencias circulares: inyectando un proxy perezoso, ambos beans se crean sin bloquearse.
 Trampa: los errores de configuración aparecen tarde (en runtime, no al arrancar).

INTERMEDIO

SPRING BOOT

Núcleo IoC

S015

RESPUESTA

BFPF: opera sobre DEFINICIONES de beans ANTES de instanciarlos (modificar propiedades, registrar más beans).
 BPP: envuelve cada bean DESPUÉS de instanciar (`postProcessBeforeInitialization/After`) — aquí nacen los proxies de `@Transactional`, `@Async`, `@Validated`, `@Lazy`.
 Es la extensibilidad del contenedor.

AVANZADO

SPRING BOOT

Núcleo IoC

S014

RESPUESTA

3 niveles de caché (`singletonFactories`, `earlySingletonReferences`, `singletons`): expone una referencia TEMPRANA del bean mientras se crea el otro.
 Solo funciona con inyección por SETTER/campo y scopes singleton; con inyección por CONSTRUCTOR falla al arrancar (y está bien: es un smell).
 Fix moderno: `@Lazy` en una dependencia o rediseñar (extraer el código común).

AVANZADO

SPRING BOOT

Núcleo IoC

S017

RESPUESTA

Problema: un singleton con dependencia prototype la recibe UNA vez.
`@Lookup`: método abstracto que Spring sobrescribe para dar una instancia nueva por llamada.
`ObjectProvider<T>`: `provider.getObject()` por petición, `getIfAvailable/getIfUnique` — más moderno y testeable.

AVANZADO

SPRING BOOT

Núcleo IoC

S016

RESPUESTA

Un bean que FABRICA otro tipo de objeto: `getObject()` devuelve la instancia real, `getObjectType` su clase, `isSingleton` su alcance.
 Uso: creación compleja de objetos de terceros (`MyBatis SqlSessionFactoryBean`, `feign clients`).
 Referencia con `&` prefijo obtiene la fábrica misma: `context.getBean("&miFactoryBean")`.

AVANZADO

SPRING BOOT

Núcleo IoC

S018

Interfaces Aware: qué inyectan

AVANZADO

SPRING BOOT

Núcleo IoC

S019

¿Cómo registras beans SIN anotaciones ni escaneo?

AVANZADO

SPRING BOOT

★

Configuración

S020

application.properties, YAML y profiles

BASICO

SPRING BOOT

★

Configuración

S021

@ConfigurationProperties vs @Value: cuál y por qué

BASICO

SPRING BOOT

Configuración

S022

Relaxed binding: ¿app.max-connections mapea a qué campo?

BASICO

SPRING BOOT

Configuración

S023

Orden de precedencia de la configuración en Spring Boot

BASICO

SPRING BOOT

Configuración

S024

¿Cómo validar que la configuración externa sea correcta al arrancar?

BASICO

SPRING BOOT

Configuración

S025

Placeholders, defaults y referencias entre propiedades

BASICO

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

SPRING BOOT

Núcleo IoC

S019

RESPUESTA

- 1) @Configuration + métodos @Bean.
- 2) ImportSelector / ImportBeanDefinitionRegistrar (mecanismo de @Import — así funcionan @Enable*).
- 3) BeanDefinitionRegistryPostProcessor programático.
- 4) spring.factories / AutoConfiguration.imports para autoconfigs. Es lo que hacen las librerías para registrar sus beans.

AVANZADO

SPRING BOOT

Núcleo IoC

S018

RESPUESTA

Callbacks para 'conciencia' del entorno: BeanNameAware (nombre del bean), ApplicationContextAware (el contexto), EnvironmentAware, ApplicationEventPublisherAware, BeanFactoryAware.

Se llaman tras la inyección de dependencias, antes de @PostConstruct.

Hoy prefieres inyección directa o implementar solo lo necesario.

AVANZADO

SPRING BOOT ★

Configuración

S021

RESPUESTA

@ConfigurationProperties(prefix="app"): POJO con muchos valores agrupados, type-safe, IDE-friendly, validable con @Validated+JSR-303, relaxed binding (app.max-retry → maxRetry).

@Value("\${clave}") : valores sueltos y rápidos, con SpEL, pero disperso y sin metadatos.

Regla: grupos de config → @ConfigurationProperties; un dato puntual → @Value.

BASICO

SPRING BOOT ★

Configuración

S020

RESPUESTA

Config externa en application.properties / .yml.

Profiles: application-prod.yml + spring.profiles.active=prod; activan beans con @Profile.

Precedencia: variables de entorno y args de CLI > perfiles > application.yml base.

BASICO

SPRING BOOT

Configuración

S023

RESPUESTA

De mayor a menor: argumentos CLI (--server.port=9090) > atributos SPRING_APPLICATION_JSON > variables de entorno (SERVER_PORT) > application-{profile} > application.yml/properties (externo > interno al jar) > @PropertySource > defaults.

Dev/test sobrescriben sin recompilar.

BASICO

SPRING BOOT

Configuración

S022

RESPUESTA

A maxConnections. Formatos equivalentes: camelCase (maxConnections), kebab-case (max-connections, recomendado en YAML), snake_case (MAX_CONNECTIONS en env vars), UPPER_CASE env var APP_MAXCONNECTIONS.

Listas: app.servers[0].host=...

Es el puente entre propiedades y variables de entorno de K8s/Cloud Run.

BASICO

SPRING BOOT

Configuración

S025

RESPUESTA

@Value("\${app.timeout:30}") — default tras los ':'. Referencias: app.url=http://\${app.host}:\${app.port}/api.

Variables especiales: \${random.value}, \${random.int[1024,65536]} (puertos de test).

SpEL en @Value: #{systemProperties['x']}.

BASICO

SPRING BOOT

Configuración

S024

RESPUESTA

@ConfigurationProperties + @Validated + anotaciones JSR-303 en los campos (@NotNull, @Min, @URL).

Configuración inválida → falla el ARRANQUE con mensaje claro, no un NPE a las 3 a.m.

Alternativa: checks manuales en @PostConstruct o fail-fast con ApplicationContextRunner en tests.

BASICO

SPRING BOOT

Configuración

S026

@Profile: activar beans por entorno

BASICO

SPRING BOOT

Configuración

S027

spring.config.import y multi-documento
YAML

BASICO

SPRING BOOT

Configuración

S028

@PropertySource y conversiones de tipos
custom

BASICO

SPRING BOOT

Configuración

S029

SpEL (Spring Expression Language): qué
permite

INTERMEDIO

SPRING BOOT

★

Boot y starters

S030

¿Qué es Spring Boot y qué aporta?

BASICO

SPRING BOOT

Boot y starters

S031

¿Qué es un starter y cuáles usas siempre?

BASICO

SPRING BOOT

Boot y starters

S032

Servidor embebido: configuración y swap

BASICO

SPRING BOOT

★

Boot y starters

S033

¿Qué hace @SpringBootApplication
exactamente?

BASICO

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

SPRING BOOT

Configuración

S027

RESPUESTA

spring.config.import=optional:file:env[.properties],
 classpath:secrets.yml — importa configs adicionales (secrets, docker).
 YAML multi-documento: '---' separa bloques, cada uno con
 spring.config.activate.on-profile — un solo archivo para todos los
 perfiles.
 Orden: el último gana.

BASICO

SPRING BOOT

Configuración

S026

RESPUESTA

@Profile("prod") / @Profile({"dev","test"}) / @Profile("!prod") en
 @Component o @Bean.
 Activar: spring.profiles.active=prod (propiedad, env var
 SPRING_PROFILES_ACTIVE, o CLI).
 Combinar con application-{profile}.yml para configuración por
 entorno.
 Un perfil default existe si no se activa ninguno.

BASICO

SPRING BOOT

Configuración

S029

RESPUESTA

Lenguaje de expresiones para valores dinámicos en tiempo de
 ejecución:
 @Value("#{systemProperties['user.name']}") — con #{...} (SpEL) vs
 \${...} (properties).
 Operadores, llamadas a métodos, colecciones: #{lista.[edad >
 18]} (selección), #{objeto.nombre?.toUpperCase()}.
 Usos: @PreAuthorize("#id == principal.id"),
 @Cacheable(key="#id"), @EventListener(condition="...").
 Poderoso pero NO metas lógica de negocio en SpEL.

INTERMEDIO

SPRING BOOT

Configuración

S028

RESPUESTA

@PropertySource("classpath:app.properties") añade fuentes al
 Environment (no soporta YAML directamente).
 Conversiones: Spring usa ConversionService — String→enum,
 duraciones ("30s" → Duration), tamaños ("10MB" → DataSize)
 automáticamente en @ConfigurationProperties; registrar
 CustomConverter para tipos propios.

BASICO

SPRING BOOT

Boot y starters

S031

RESPUESTA

Dependencia 'todo en uno' que agrupa librerías compatibles:
 spring-boot-starter-web (MVC+Tomcat+Jackson+validación),
 starter-data-jpa, starter-security, starter-actuator, starter-test,
 starter-validation.
 Eliminan versiones incompatibles: el BOM de Boot gobierna
 versiones transitivas.

BASICO

SPRING BOOT

Boot y starters

S030

RESPUESTA

Capa opinionada sobre Spring que elimina la configuración
 manual:

- Autoconfiguración según el classpath
- Starters (spring-boot-starter-web)
- Servidor embebido (Tomcat/Jetty)
- Actuator listo para producción

@SpringBootApplication = @Configuration +
 @EnableAutoConfiguration + @ComponentScan.

BASICO

SPRING BOOT

Boot y starters

S033

RESPUESTA

Agrupar 3: @SpringBootApplication (es una @Configuration),
 @EnableAutoConfiguration y @ComponentScan (paquete actual
 hacia abajo).
 Atributos útiles: exclude=DataSourceAutoConfiguration.class
 (quita autoconfigs), scanBasePackages (mover el escaneo).
 Debe vivir en el paquete raíz.

BASICO

SPRING BOOT

Boot y starters

S032

RESPUESTA

Tomcat por defecto; starter-web-reactor trae Netty (WebFlux), y
 puedes cambiar a Jetty/Undertow excluyendo starter-tomcat.
 Config: server.port, server.servlet.context-path,
 server.tomcat.threads.max, graceful shutdown
 (server.shutdown=graceful).
 Jar ejecutable con java -jar — sin instalar servlet container.

BASICO

SPRING BOOT

Boot y starters

S034

Anatomía de una autoconfiguración (condiciones)

BASICO

SPRING BOOT

Boot y starters

S035

Cómo depurar la autoconfiguración (por qué se activó X)

BASICO

SPRING BOOT

Boot y starters

S036

Clase main: SpringApplication y run() qué devuelve

BASICO

SPRING BOOT

Boot y starters

S037

Baner, nombres de app y metadata útil

BASICO

SPRING BOOT

Boot y starters

S038

Excluir autoconfiguración: las 3 formas

BASICO

SPRING BOOT

Boot y starters

S039

Propiedades comunes de arranque que conviene saber de memoria

BASICO

SPRING BOOT

Boot y starters

S040

Spring Boot DevTools: qué aporta en desarrollo

BASICO

SPRING BOOT

Boot y starters

S041

¿Cómo funciona la autoconfiguración?

INTERMEDIO

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

SPRING BOOT

Boot y starters

S035

RESPUESTA

--debug o debug=true → CONDITION EVALUATION REPORT al arrancar: cada autoconfig con positivo/negativo y POR QUÉ (conditional on class, missing bean, property).
También: actuador/conditions endpoint.
Es el primer paso ante 'no se configura mi DataSource/conector'.

BASICO

SPRING BOOT

Boot y starters

S034

RESPUESTA

@AutoConfiguration +
@ConditionalOnClass(DataSource.class) — si la clase está en el classpath, activa.
@ConditionalOnMissingBean: si TÚ ya definiste uno, respeta el tuyo.
@ConditionalOnProperty(prefix="spring.datasource", name="url").
Listadas en META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports.

BASICO

SPRING BOOT

Boot y starters

S037

RESPUESTA

spring.application.name: identificador en logs/tracing/eureka.
banner.txt custom o off.
build-info: spring-boot-maven-plugin con build-info → /actuador/info muestra versión y timestamps.
Los metadatos de propiedades (additional-spring-configuration-metadata.json) dan autocomplete y documentación en el IDE.

BASICO

SPRING BOOT

Boot y starters

S036

RESPUESTA

ApplicationContext context = SpringApplication.run(App.class, args).
Obtener beans: context.getBean(...), publicar eventos, perfiles activos.
Customizar: new SpringApplication(App.class).bannerMode(OFF).profiles("x").run(args).
CommandLineRunner/ApplicationRunner: código de arranque post-contexto (reciben args crudos/parsed).

BASICO

SPRING BOOT

Boot y starters

S039

RESPUESTA

server.port=0 (puerto aleatorio para tests), spring.profiles.active,
spring.datasource.url/username/password,
spring.jpa.hibernate.ddl-auto (none/validate/update/create),
spring.jpa.show-sql, management.endpoints...,
logging.level.package=DEBUG,
spring.servlet.multipart.max-file-size.
En YAML con kebab-case.

BASICO

SPRING BOOT

Boot y starters

S038

RESPUESTA

1) @SpringBootApplication(exclude = {DataSourceAutoConfiguration.class}).
2) spring.autoconfigure.exclude=... (propiedad, soporta lista).
3) Definir TÚ el bean: @ConditionalOnMissingBean cede (no es exclusión, pero anula el efecto).
Caso típico: BD en memoria de Boot cuando la quieres manual.

BASICO

SPRING BOOT



Boot y starters

S041

RESPUESTA

@EnableAutoConfiguration carga configuraciones candidatas que se activan SOLO si se cumplen condiciones:
@ConditionalOnClass, @ConditionalOnMissingBean, @ConditionalOnProperty.
Ej.: detecta H2 en el classpath → configura un DataSource.
Se excluye con exclude=... en @SpringBootApplication.

INTERMEDIO

SPRING BOOT

Boot y starters

S040

RESPUESTA

Reinicio RÁPIDO al guardar (restart con classloader separado: tus clases recargan, las libs no → segundos vs minutos).
LiveReload integrado para frontends.
Deshabilita cachés por defecto en dev.
Se desactiva SOLO al empaquetar (jar de producción no lo lleva).
spring-boot-devtools en scope runtime (optional).

BASICO

SPRING BOOT ★

Web MVC y REST

S042

@Controller vs @RestController

BASICO

SPRING BOOT ★

Web MVC y REST

S043

@PathVariable vs @RequestParam vs @RequestBody

BASICO

SPRING BOOT ★

Web MVC y REST

S044

¿Cómo manejas excepciones globalmente en una API?

BASICO

SPRING BOOT

Web MVC y REST

S045

Anotaciones de mapeo: combinadas y específicas

BASICO

SPRING BOOT

Web MVC y REST

S046

ResponseEntity: cuándo y cómo

BASICO

SPRING BOOT ★

Web MVC y REST

S047

Jackson: anotaciones que resuelven el 90% de los casos

BASICO

SPRING BOOT ★

Web MVC y REST

S048

¿Por qué NUNCA devolver entidades JPA en un controller?

BASICO

SPRING BOOT ★

Web MVC y REST

S049

Validación: @Valid, @Validated y constraints

BASICO

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

SPRING BOOT



Web MVC y REST

S043

RESPUESTA

@PathVariable: del path → GET /users/{id}
 @RequestParam: query string → ?page=2&size=10 (defaultValue, required)
 @RequestBody: deserializa el body (JSON → DTO) en POST/PUT.
 @RequestHeader para cabeceras.

BASICO

SPRING BOOT



Web MVC y REST

S042

RESPUESTA

@Controller: devuelve nombres de VISTA (HTML con Thymeleaf, etc.).
 @RestController: añade @ResponseBody → cada método retorna el objeto que Spring serializa a JSON con HttpMessageConverters (Jackson).
 Para APIs REST: siempre @RestController.

BASICO

SPRING BOOT

Web MVC y REST

S045

RESPUESTA

@RequestMapping(method=...) → atajos: @GetMapping, @PostMapping, @PutMapping, @PatchMapping, @DeleteMapping.
 Atributos: path/value, params (condición por query param), headers, produces/consumes (negociación).
 Class-level + method-level se CONCATENAN los paths.

BASICO

SPRING BOOT



Web MVC y REST

S044

RESPUESTA

@RestControllerAdvice con métodos
 @ExceptionHandler(Exception.class): mapean excepciones de toda la app a respuestas HTTP uniformes (status + cuerpo de error).
 Spring 6 trae ProblemDetail (RFC 7807).
 @ResponseStatus para casos simples.

BASICO

SPRING BOOT



Web MVC y REST

S047

RESPUESTA

@JsonProperty("nombre_json") renombra; @JsonIgnore oculta;
 @JsonFormat(shape/pattern, timezone) para fechas;
 @JsonInclude(NON_NULL) evita nulls; @JsonAlias acepta varios nombres al deserializar; @JsonCreator/@JsonProperty para constructores inmutables;
 @JsonManagedReference/@JsonBackReference corta recursión en relaciones bidireccionales (o mejor: usa DTOs).

BASICO

SPRING BOOT

Web MVC y REST

S046

RESPUESTA

Control total de status + headers + body: ResponseEntity.status(HttpStatus.CREATED).location(uri).body(dto).
 Uso típico: 201 con Location en POST, 204 en DELETE, 304 con etag.
 Alternativa moderna: devolver el DTO directo + @ResponseStatus, o ProblemDetail en excepciones.

BASICO

SPRING BOOT



Web MVC y REST

S049

RESPUESTA

@Valid en @RequestBody dispara Bean Validation (Hibernate Validator): @NotNull, @NotBlank, @Size, @Min/@Max, @Email, @Pattern, @Positive, @Past/Future.
 Fallo → MethodArgumentNotValidException → 400 con detalles (capturar en @ControllerAdvice).
 @Validated en clase: valida parámetros de métodos simples (y habilita groups).
 Custom: @Constraint + ConstraintValidator.

BASICO

SPRING BOOT



Web MVC y REST

S048

RESPUESTA

- 1) LazyInitializationException o SELECTs N+1 al serializar relaciones lazy.
- 2) Recursión infinita en relaciones bidireccionales.
- 3) Expone tu esquema de BD (campos sensibles, breaking changes).
- 4) Acopla API y dominio.

Fix: DTOs + MapStruct/constructor; relaciones fetcheadas explícitas.

BASICO

SPRING BOOT

Web MVC y REST

S050

Interceptor vs Filter vs AOP: ¿cuál para cada caso?

BASICO

SPRING BOOT

Web MVC y REST

S051

HandlerInterceptor: implementación y registro

BASICO

SPRING BOOT

Web MVC y REST

S052

CORS: las 3 formas de configurarlo en Spring

BASICO

SPRING BOOT

Web MVC y REST

S053

Paginación y ordenación estándar en Spring

BASICO

SPRING BOOT

Web MVC y REST

S054

Upload de archivos (multipart) en Spring

BASICO

SPRING BOOT

Web MVC y REST

S055

Manejo de errores REST moderno: ProblemDetail

BASICO

SPRING BOOT

Web MVC y REST

S056

@ExceptionHandler jerarquía: quién gana

BASICO

SPRING BOOT

Web MVC y REST

S057

Clientes HTTP modernos: RestClient vs WebClient vs Feign

BASICO

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

SPRING BOOT

Web MVC y REST

S051

RESPUESTA

implements HandlerInterceptor { preHandle (puede abortar con false), postHandle (antes de renderizar body), afterCompletion (siempre, para cleanup).
 Registro: WebMvcConfigurer addInterceptors → registry.addInterceptor(miInterceptor).addPathPatterns("/api/**").excludePathPatterns("/actuator/**").

BASICO

SPRING BOOT

Web MVC y REST

S050

RESPUESTA

Filter (Servlet): ANTES de que llegue a Spring — auth cruda, compresión, charset, wrapping de request.
 HandlerInterceptor: dentro de Spring MVC — preHandle/postHandle/afterCompletion, acceso a handler elegido: logging de requests, métricas, tenant resolution.
 AOP: capa de servicio/dominio — transacciones, caché, auditoría de negocio.
 Orden: Filters → DispatcherServlet → Interceptors → Controller → AOP.

BASICO

SPRING BOOT

Web MVC y REST

S053

RESPUESTA

Método recibe Pageable (Spring resuelve ?page=0&size=20&sort=nombre,asc) → devuelve Page<Dto> (content + totalElements + totalPages).
 Customizar nombres de parámetros: spring.data.web.pageable.*.
 Límite duro para size (máx 100) — evita 'page size 1 millón'.
 En DTO de salida: exponer solo lo necesario o devolver Slice (¿hay más?) si no quieres COUNT.

BASICO

SPRING BOOT

Web MVC y REST

S052

RESPUESTA

1) @CrossOrigin(origins, methods) por controller/método.
 2) WebMvcConfigurer.addCorsMappings: registry.addMapping("/api/**").allowedOrigins(...).allowedMethods(...).
 3) Con Security: http.cors(Customizer) + CorsConfigurationSource bean (este manda).
 Preflight OPTIONS: Spring responde automáticamente si el origen está permitido.

BASICO

SPRING BOOT

Web MVC y REST

S055

RESPUESTA

Spring 6+: @ExceptionHandler puede devolver ProblemDetail (RFC 9457): { type, title, status, detail, properties... }.
 Autoconfigurable: spring.mvc.problemdetails.enabled=true.
 Cliente recibe error MACHINE-READABLE y humano, estándar entre APIs.
 En @RestControllerAdvice mapear cada excepción de negocio a su status.

BASICO

SPRING BOOT

Web MVC y REST

S054

RESPUESTA

@PostMapping(consume = MediaType.MULTIPART_FORM_DATA_VALUE) con @RequestParam("file") MultipartFile file.
 Config: spring.servlet.multipart.max-file-size / max-request-size.
 Guardado: Files.copy(file.getInputStream(), destino) — nunca en memoria para grandes.
 Validar content-type y extensión (no confiar en el filename del cliente).

BASICO

SPRING BOOT

Web MVC y REST

S057

RESPUESTA

RestClient (Spring 6.1): API fluida SÍNCRONA, el reemplazo natural de RestTemplate.
 WebClient: reactivo (o síncrono bloqueando), necesario en WebFlux.
 RestTemplate: en mantenimiento, no para código nuevo.
 OpenFeign: interfaz declarativa + anotaciones — genera la implementación (común en microservicios con Spring Cloud).

BASICO

SPRING BOOT

Web MVC y REST

S056

RESPUESTA

El más ESPECÍFICO y más CERCANO gana: método del controller > @ControllerAdvice > handler por defecto (BasicExceptionHandler).
 Varios advices: @Order decide.
 Patrón: un advice global con ExceptionHandler para BusinessException, ValidationException, AccessDenied, y un catch-all 500 que loguea con stack.

BASICO

SPRING BOOT

Web MVC y REST

S058

RestClient: ejemplo canónico

BASICO

SPRING BOOT

Web MVC y REST

S059

DTOs: mapeo manual vs MapStruct

BASICO

SPRING BOOT

Web MVC y REST

S060

Arquitectura en capas vs hexagonal dentro de un servicio Spring

BASICO

SPRING BOOT

Web MVC y REST

S061

¿Dónde va la lógica de negocio: controller, service o repository?

BASICO

SPRING BOOT

Web MVC y REST

S062

¿Cómo manejar ids en URIs y respuestas de creación?

BASICO

SPRING BOOT

Web MVC y REST

S063

Multipart + DTO combinados en un endpoint

BASICO

SPRING BOOT

★

Web MVC y REST

S064

Anatomía de una petición HTTP (lo básico que nunca está de más)

BASICO

SPRING BOOT

★

Web MVC y REST

S065

Flujo de una petición en Spring MVC

INTERMEDIO

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

SPRING BOOT

Web MVC y REST

S059

RESPUESTA

Manual: constructor/copiar a mano — explícito, cero magia, verboso.
 MapStruct: genera el mapper EN COMPILACIÓN (sin reflejos, rápido): @Mapper interface con métodos; mapea por nombres, customiza con @Mapping(source, target, ignore).
 ModelMapper: runtime + reflejo — cómodo pero silencioso ante refactor.
 Elige: MapStruct para proyectos serios.

BASICO

SPRING BOOT

Web MVC y REST

S058

RESPUESTA

```
RestClient rc = RestClient.create();
Pedido p = rc.get().uri("http://svc/api/pedidos/{id}",
id).retrieve().body(Pedido.class);
POST: rc.post().uri(...).contentType(APPLICATION_JSON).body(dto)
.retrieve().toBodilessEntity();
Errores: .onStatus(HttpStatus::isError, (req,res) -> { throw
new MiEx(); })
Builder con baseUrl para reutilizar.
```

BASICO

SPRING BOOT

Web MVC y REST

S061

RESPUESTA

Controller: HTTP puro — validar formato, mapear, responder.
 Service: REGLAS DE NEGOCIO, orquestación, transaccionalidad.
 Repository: acceso a datos únicamente (queries), sin lógica.
 Anemia (services con if/else y entidades tontas) es común pero mejorable: lógica en entidades de dominio cuando pertenece a un solo agregado.

BASICO

SPRING BOOT

Web MVC y REST

S060

RESPUESTA

Capas clásica: controller → service → repository, DTOs en la orilla. Simple y conocida.
 Hexagonal: dominio en el CENTRO sin dependencias de Spring/JPA; puertos (interfaces) + adapters (web, persistencia) inyectados.
 Beneficio hexagonal: dominio testeable sin levantar Spring, swap de infraestructura.
 Costo: más código; justificado en dominios complejos.

BASICO

SPRING BOOT

Web MVC y REST

S063

RESPUESTA

@RequestParam("datos") MiDto dto + @RequestParam("archivo")
 MultipartFile file — cada parte con su propio
 converter/validación (@Valid sobre la parte).
 Consumes MULTIPART_FORM_DATA. El cliente envía form-data
 con partes tipadas.
 Caso típico: metadata JSON + archivo.

BASICO

SPRING BOOT

Web MVC y REST

S062

RESPUESTA

POST → 201 + header Location: /api/pedidos/42 (URI absoluta o relativa): ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(dto.getId()).toUri().
 Body: el recurso creado (o solo id).
 Ids: no exponer secuenciales de BD si importan (UUID o ids opacos).

BASICO

SPRING BOOT ★

Web MVC y REST

S065

RESPUESTA

Request → DispatcherServlet (front controller) →
 HandlerMapping elige el método → interceptores → argumentos
 resueltos (@PathVariable, @RequestBody...) → tu método →
 HttpMessageConverter serializa → respuesta.
 Excepciones → @ExceptionHandler / @ControllerAdvice.

INTERMEDIO

SPRING BOOT ★

Web MVC y REST

S064

RESPUESTA

Request: MÉTODO (GET/POST/PUT...) + PATH (/api/users/1) +
 HEADERS (Content-Type, Authorization, Accept) + BODY
 opcional.
 Response: STATUS (200/404/500) + HEADERS + BODY.
 Sin estado: cada request se entiende sola; la 'sesión' viaja en
 tokens o cookies.
 En Spring: método=GetMapping, path=RequestMapping,
 headers=@RequestHeader, body=@RequestBody,
 status=ResponseEntity.

BASICO

SPRING BOOT

Web MVC y REST

S066

@ModelAttribute y formularios clásicos vs @RequestBody

INTERMEDIO

SPRING BOOT

Web MVC y REST

S067

Versión de API: las 3 estrategias aplicadas en Spring

INTERMEDIO

SPRING BOOT

Web MVC y REST

S068

Documentar la API: OpenAPI/Swagger en Spring Boot 3

INTERMEDIO

SPRING BOOT

Web MVC y REST

S069

GlobalExceptionHandler: esqueleto que debes saber escribir

INTERMEDIO

SPRING BOOT

Web MVC y REST

S070

Content negotiation: cómo Spring elige JSON o XML

AVANZADO

SPRING BOOT

Web MVC y REST

S071

Grupos de validación: crear vs actualizar

AVANZADO

SPRING BOOT

Web MVC y REST

S072

ConstraintValidator: validación custom paso a paso

AVANZADO

SPRING BOOT

Web MVC y REST

S073

ETags y caché HTTP con Spring

AVANZADO

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

SPRING BOOT

Web MVC y REST

S067

RESPUESTA

URI: @GetMapping("/api/v1/pedidos") + controllers por versión (clara, cacheable, la más común).
 Header: @GetMapping(headers="X-API-Version=2") o Accept versionado — URL limpia pero menos visible.
 Query param: ?version=2 (flexible, no estándar).
 Regla: no romper contratos; versiones nuevas = código nuevo coexistiendo, deprecarse con aviso.

INTERMEDIO

SPRING BOOT

Web MVC y REST

S066

RESPUESTA

@RequestBody: body JSON/XML → DTO (via `HttpMessageConverter`). APIs.
 @ModelAttribute: binding de form-data/query a objeto con setters + validación — apps MVC con vistas, formularios HTML.
 Errores de binding de form: `BindingResult` inmediatamente tras el parámetro para manejarlos sin excepción.

INTERMEDIO

SPRING BOOT

Web MVC y REST

S069

RESPUESTA

@RestControllerAdvice class:
 @ExceptionHandler(RecursoNoEncontrado.class) → 404 `ProblemDetail`.
 @ExceptionHandler(MethodArgumentNotValidException.class) → 400 + lista de campos inválidos (`ex.getBindingResult().getFieldErrors()`).
 @ExceptionHandler(AccesoDenegado.class) → 403.
 @ExceptionHandler(Exception.class) → 500 + `log.error` con stack (sin exponer detalles internos al cliente).

INTERMEDIO

SPRING BOOT

Web MVC y REST

S068

RESPUESTA

springdoc-openapi-starter-webmvc-ui: expone `/v3/api-docs` (JSON OpenAPI) y `/swagger-ui.html`.
 Enriquecer: @Operation(summary, description), @ApiResponse, @Schema en DTOs, @Tag por controller.
 Exportar spec para contratos/gateways. Documentar es parte del contrato — pruebas de contrato la validan.

INTERMEDIO

SPRING BOOT

Web MVC y REST

S071

RESPUESTA

POST requiere algunos campos, PUT no (id autogenerado).
 public interface OnCreate {} y @NotNull(groups=OnCreate.class) Long id.
 @Validated(OnCreate.class) en el POST.
 Alternativa moderna: DTOs separados por operación (`CreateUserRequest/UpdateUserRequest`) — más claro que grupos.

AVANZADO

SPRING BOOT

Web MVC y REST

S070

RESPUESTA

Por headers Accept (y Content-Type para el body de entrada) contra los produce/consume declarados y los `HttpMessageConverters` disponibles.
 Jackson (JSON) viene con starter-web.
 URL con sufijo (.json) está DESACTIVADO por defecto (seguridad).
 Formato param: favor-parameter (raro).

AVANZADO

SPRING BOOT

Web MVC y REST

S073

RESPUESTA

`ResponseEntity` con eTag + condicionales:
`ResponseEntity.ok().eTag(version).body(dto)`; manejo de If-None-Match → 304.
`ShallowETagHeaderFilter` (hash del body) o eTag manual por versión de entidad (@Version).
 Cache-Control header + `CacheControl.maxAge(...)`.
 Reduce ancho de banda; no sustituye caché de aplicación.

AVANZADO

SPRING BOOT

Web MVC y REST

S072

RESPUESTA

1) @interface @Target(FIELD) @Constraint(validatedBy = EstadoValidator.class).
 2) class EstadoValidator implements ConstraintValidator<MiAnon, String> { isValid(value, ctx) }.
 3) `ctx.buildConstraintViolationWithTemplate(msg)` para mensajes dinámicos.
 Se aplica como cualquier constraint; parametrizable con atributos de la anotación.

AVANZADO

SPRING BOOT

Web MVC y REST

S074

Streaming de respuestas: grandes exportaciones

AVANZADO

SPRING BOOT

Web MVC y REST

S075

Async requests MVC: DeferredResult y Callable

AVANZADO

SPRING BOOT

Web MVC y REST

S076

HATEOAS: qué es y cuándo (no) usarlo

AVANZADO

SPRING BOOT

★Spring Data JPA

S077

Query derivation: keywords que se preguntan

BASICO

SPRING BOOT

★Spring Data JPA

S078

@Query: JPQL vs nativo y cuándo cada uno

BASICO

SPRING BOOT

Spring Data JPA

S079

Proyecciones: interface-based y class-based

BASICO

SPRING BOOT

★Spring Data JPA

S080

save() vs saveAndFlush() vs merge

BASICO

SPRING BOOT

★Spring Data JPA

S081

Relaciones JPA: anotaciones y el lado dueño

BASICO

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

SPRING BOOT

Web MVC y REST

S075

RESPUESTA

Devolver Callable<T> o DeferredResult<T> libera el servlet thread y responde cuando la tarea termina — más throughput con I/O lenta sin migrar a WebFlux.

Timeouts configurables; CompletableFuture es la base moderna. StreamingResponseBody para escribir progresivamente. WebFlux es otra escala (reactivo completo), no siempre necesario.

AVANZADO

SPRING BOOT

Web MVC y REST

S074

RESPUESTA

SseEmitter (eventos), StreamingResponseBody (escribe en el OutputStream en otro hilo — no retiene el servlet thread con todo el body), ResponseEntity<Resource> con InputStreamResource para archivos.

Para exportación de reportes: escribir CSV/Excel por chunks directamente al stream.

Evita OOM al exportar millones de filas.

AVANZADO

SPRING BOOT



Spring Data JPA

S077

RESPUESTA

findByEmailAndActivo(email, bool),
findByNombreStartingWith/Containing/EndingWith, Between, IsNull/IsNotNull, In, OrderByEdadDesc (en el nombre), Top5/First3 (limit), findByEdadGreaterThanOrEqualTo.
Conteo: countBy...; existencia: existsBy...; borrado: deleteBy... (requiere @Transactional y carga entidades).
Si el nombre crece demasiado → @Query.

BASICO

SPRING BOOT

Web MVC y REST

S076

RESPUESTA

Hypermedia: la respuesta incluye links a acciones posibles (_links.self, .cancel) — el cliente navega sin conocer URLs fijas.

Spring HATEOAS: EntityModel/WebMvcLinkBuilder.

Realismo: rara vez se implementa completo (nivel 3 Richardson); valor real en APIs públicas longevas. Internamente, links simples bastan.

AVANZADO

SPRING BOOT

Spring Data JPA

S079

RESPUESTA

Interface projection: interface NombreVista { String getNombre(); BigDecimal getTotal(); } — el repo devuelve List<NombreVista>; Spring genera proxy que solo trae esas columnas (SQL optimizado).

Class/DTO projection: @Query con 'select new com.x.Dto(p.nombre, sum(p.total))'.

Dynamic: <T> List<T> findBy...(Class<T> tipo).
Evita cargar entidades para solo leer columnas.

BASICO

SPRING BOOT



Spring Data JPA

S078

RESPUESTA

@Query("select p from Pedido p where p.total > :min") — JPQL contra ENTIDADES (portable, tipo-aware con @Param).

@Query(value="select * from pedidos...", nativeQuery=true) — SQL real: funciones de BD, hints, UNION; acopla al motor.

Retornos posibles: entidades, DTOs con constructor expression (select new com.x.Dto(p.id, p.total)), proyecciones de interfaz, scalars.

BASICO

SPRING BOOT



Spring Data JPA

S081

RESPUESTA

@OneToOne, @OneToMany/@ManyToOne, @ManyToMany.

El lado DUEÑO tiene la FK: @JoinColumn("cliente_id") en ManyToOne.

@OneToMany(mappedBy="cliente"): lado inverso (no dueño) — sin tabla intermedia propia.

En @ManyToMany ALWAYS solo un lado mappedBy + @JoinTable; cuida cascade y orphanRemoval.

BASICO

SPRING BOOT



Spring Data JPA

S080

RESPUESTA

save: si es nueva → persist; si existe (no transient) → merge; el INSERT/UPDATE puede diferirse al flush.

saveAndFlush: fuerza el SQL YA (necesitas el id generado o triggers al instante).

merge: copia el estado de una entidad detached a una managed (devuelve la MANAGED — ignorar el retorno es un bug clásico).

BASICO

SPRING BOOT

★

Spring Data JPA

S082

Cascade: tipos y cuándo usarlos

BASICO

SPRING BOOT

Spring Data JPA

S083

@Embeddable y @Embedded: value objects en entidades

BASICO

SPRING BOOT

Spring Data JPA

S084

AttributeConverter: mapear enums/objetos custom

BASICO

SPRING BOOT

Spring Data JPA

S085

Repositorios múltiples y @NoRepositoryBean

BASICO

SPRING BOOT

★

Spring Data JPA

S086

¿Qué es JPA? ¿Y Hibernate? ¿Y Spring Data JPA?

INTERMEDIO

SPRING BOOT

★

Spring Data JPA

S087

Estados de una entidad JPA

INTERMEDIO

SPRING BOOT

★

Spring Data JPA

S088

¿Qué es LazyInitializationException y cómo se evita?

INTERMEDIO

SPRING BOOT

★

Spring Data JPA

S089

JOIN FETCH y @EntityGraph: resolver N+1

INTERMEDIO

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

SPRING BOOT

Spring Data JPA

S083

RESPUESTA

Clase @Embeddable Direccion (calle, ciudad, cp) embebida en @Embedded Direccion direccion → columnas en la MISMA tabla.
 @AttributeOverrides renombra columnas si se repite.
 Perfecto para value objects inmutables con equals/hashCode (dinero, dirección, rango de fechas).

BASICO

SPRING BOOT



Spring Data JPA

S082

RESPUESTA

PERSIST (guardar hijos con el padre), MERGE, REMOVE (borrar hijos con el padre — cuidado), ALL, REFRESH, DETACH.
 Uso: ciclo de vida COMPLETAMENTE dependiente (Pedido→Items: ALL + orphanRemoval=true).
 NUNCA cascade a entidades compartidas (no pongas ALL de Pedido→Cliente: borrar un pedido no debe borrar un cliente).

BASICO

SPRING BOOT

Spring Data JPA

S085

RESPUESTA

Base repository compartido: interface BaseRepo<T> extends JpaRepository<T, ID> con métodos comunes — anótala @NoRepositoryBean para que Spring NO la instancie.
 Cada entidad extiende la base. También exist: fragmentos custom (interface custom + Impl) para lógica de acceso compartida.

BASICO

SPRING BOOT

Spring Data JPA

S084

RESPUESTA

@Converter(autoApply=true) class RolConverter implements AttributeConverter<Rol, String> { convertToDatabaseColumn / convertToEntityAttribute }.
 Casos: enums por código legado, clases de valor (Dinero→2 columnas via Embeddable mejor), listas como JSON.
 Aplica a todas las entidades sin repetir.

BASICO

SPRING BOOT



Spring Data JPA

S087

RESPUESTA

Transient: nuevo, fuera del contexto.
 Managed: attached — el dirty checking detecta cambios y los sincroniza en flush (sin llamar a save).
 Detached: el contexto cerró; save() hace merge para re-adjuntar.
 Removed: marcado para DELETE.

INTERMEDIO

SPRING BOOT



Spring Data JPA

S086

RESPUESTA

JPA: especificación de ORM en Java (mapear entidades con @Entity, @Id, relaciones).
 Hibernate: implementación de referencia.
 Spring Data JPA: capa encima que genera repositorios (JpaRepository) y queries por nombre de método (findByEmail) o @Query.

INTERMEDIO

SPRING BOOT



Spring Data JPA

S089

RESPUESTA

N+1: 1 query de pedidos + N queries de cliente al tocar cada relación lazy.
 @Query("select p from Pedido p join fetch p.cliente where p.id in :ids") — fetch join trae la relación EAGER en ESA query.
 @EntityGraph(attributePaths = {"cliente", "items"}) sobre el método del repo — mismo efecto declarativo.
 Distinct + fetch de colecciones → in-memory dedup (o hibernate.default_batch_fetch_size).

INTERMEDIO

SPRING BOOT



Spring Data JPA

S088

RESPUESTA

Acceder a una relación LAZY cuando la sesión de Hibernate ya cerró.
 Soluciones:
 • JOIN FETCH en JPQL (@Query)
 • @EntityGraph
 • proyecciones a DTO
 • transacción abierta en el servicio (evita depender de open-in-view).

INTERMEDIO

SPRING BOOT

Spring Data JPA

S090

@Modifying y updates masivos

INTERMEDIO

SPRING BOOT

Spring Data JPA

S091

orphanRemoval vs CascadeType.REMOVE

INTERMEDIO

SPRING BOOT

Spring Data JPA

S092

Herencia en entidades: las 3 estrategias

INTERMEDIO

SPRING BOOT

Spring Data JPA

S093

Optimistic locking con @Version

INTERMEDIO

SPRING BOOT

Spring Data JPA

S094

Pessimistic locking: cuándo y cómo

INTERMEDIO

SPRING BOOT

Spring Data JPA

S095

Specifications (Criteria): queries dinámicas tipadas

INTERMEDIO

SPRING BOOT

Spring Data JPA

S096

Auditoría automática: @CreatedDate y compañía

INTERMEDIO

SPRING BOOT

Spring Data JPA

S097

Batching: cómo hacer que inserts/updates masivos no sean 1 a 1

INTERMEDIO

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

SPRING BOOT

Spring Data JPA

S091

RESPUESTA

orphanRemoval=true: si QUITAS un hijo de la colección del padre, se DELETE de BD (los huérfanos se eliminan).
 CascadeType.REMOVE: borrar el PADRE borra a los hijos.
 Combinación estándar en composiciones:
 @OneToMany(mappedBy=..., orphanRemoval=true, cascade=ALL).
 Orphan removal es reemplazo de hijo:
 pedido.getItems().remove(item) → DELETE.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S090

RESPUESTA

@Modifying + @Query("update Pedido p set p.estado = :e where p.creado < :fecha") — UPDATE/DELETE directo en BD.
 Requiere @Transactional.
 NO actualiza el contexto de persistencia: entidades en memoria quedan desactualizadas (clearAutomatically=true para forzar).
 Mucho más rápido que cargar+modificar miles de entidades.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S093

RESPUESTA

Campo @Version Long version en la entidad: cada UPDATE incluye WHERE version = leída y la incrementa.
 Concurrencia: segundo hilo con versión vieja → OptimisticLockException (o ObjectOptimisticLockingFailureException en Spring).
 Sin locks de BD: ideal para baja contención.
 Reintentar la operación es responsabilidad de la app.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S092

RESPUESTA

@Inheritance(strategy=...):
 SINGLE_TABLE (default): una tabla + columna discriminador — rápida, columnas nullable.
 JOINED: tabla por clase unidas por FK — normalizada, joins en cada lectura.
 TABLE_PER_CLASS: tabla completa por clase concreta — uniones costosas, ids compartidos.
 Default razonable: SINGLE_TABLE.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S095

RESPUESTA

extend JpaSpecificationExecutor<T>: findAll(Specification<T> spec, Pageable).
 Construir con Specification.where(A.and(B)): (root, query, cb) -> cb.equal(root.get("estado"), estado).
 Caso de uso: filtros de búsqueda contruidos por combinación (campos opcionales).
 Tipado con Metamodel para evitar strings.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S094

RESPUESTA

Contención alta donde reintentar es caro (reservas, saldos):
 @Lock(LockModeType.PESSIMISTIC_WRITE) en el método del repo → SELECT ... FOR UPDATE (bloquea la fila hasta el commit).
 Con timeout: @QueryHints(@QueryHint(name="javax.persistence.lock.timeout", value="3000")).
 Cuidado: orden consistente de bloqueo para evitar deadlocks.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S097

RESPUESTA

spring.jpa.properties.hibernate.jdbc.batch_size=50 + order_inserts=true + order_updates=true.
 REQUISITO con MySQL: rewriteBatchedStatements=true en la URL.
 OJO: IDs IDENTITY DESACTIVAN batching en inserts (usa SEQUENCE/SEQUENCE-style o asignados).
 Guardar en chunks de 50 con clear() periódico del contexto.

INTERMEDIO

SPRING BOOT

Spring Data JPA

S096

RESPUESTA

@EnableJpaAuditing en config + @EntityListeners(AuditingEntityListener.class).
 @CreatedDate/@LastModifiedDate (fechas), @CreatedBy/@LastModifiedBy (necesitas AuditorAware<T> bean que devuelve el usuario actual).
 Campos auditables en un MappedSuperclass base para reusar en todas las entidades.

INTERMEDIO

SPRING BOOT ★

Spring Data JPA

S098

ddl-auto: valores y cuál en cada entorno

INTERMEDIO

SPRING BOOT ★

Spring Data JPA

S099

Flyway/Liquibase: migraciones versionadas

INTERMEDIO

SPRING BOOT

Spring Data JPA

S100

Dirty checking: cómo Hibernate detecta cambios sin save()

INTERMEDIO

SPRING BOOT

Spring Data JPA

S101

flush y el orden: queries antes del commit

INTERMEDIO

SPRING BOOT

Spring Data JPA

S102

Repositorio custom: interface + Impl (el patrón completo)

INTERMEDIO

SPRING BOOT

Spring Data JPA

S103

EntityManager vs repositorio: cuándo bajar al EntityManager

INTERMEDIO

SPRING BOOT ★

Spring Data JPA

S104

JdbcTemplate: para qué sirve y por qué sigue vivo

INTERMEDIO

SPRING BOOT

Spring Data JPA

S105

Spring Data JDBC vs Spring Data JPA: cuándo cada uno

INTERMEDIO

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

SPRING BOOT ★
RESPUESTA

Spring Data JPA

S099

Scripts SQL versionados en resources/db/migration:
V1_crear_tabla.sql, V2_add_column.sql.
Al arrancar, Flyway aplica los pendientes en orden y registra en flyway_schema_history.
Beneficios: esquema reproducible, migraciones revisadas en PR, rollback claro (V_down o revert).
Regla: cambiar entidad = nueva migración, nunca editar la vieja aplicada.

INTERMEDIO

SPRING BOOT ★
RESPUESTA

Spring Data JPA

S098

create: borra y crea TODO al arrancar (solo dev).
create-drop: crea al arrancar, borra al cerrar (tests).
update: altera tablas existentes (NUNCA en prod: sin rollback ni historial).
validate: solo verifica el esquema (prod recomendado con migraciones).
Esquema real de prod: Flyway/Liquibase + validate.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S101

Flush = enviar SQL pendiente a BD (sin commit). Ocurre: en commit, en queries JPQL (para ver cambios pendientes, flush AUTO), y con flush() explícito.
Trampa: query nativa NO dispara flush de la entidad (FlushModeType.AUTO no aplica igual) → leer datos viejos; usar entityManager.flush() antes.
Commit = flush + transacción confirmada.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S100

Al cargar, Hibernate guarda snapshot del estado; al flush compara y genera UPDATE solo con columnas cambiadas.
Consecuencia: dentro de una transacción, entidad cargada + setter = UPDATE automático (sin llamar save).
Fuera de transacción (o detached) no ocurre — por eso la gente 'pierde' cambios.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S103

Repositorio: 90% de casos CRUD/queries nombradas.
EntityManager (@PersistenceContext): queries criteria dinámicas, flush/clear manuales en procesos masivos, detach para liberar memoria, batching con clear cada N, locks con hints, referencias lazy getReference.
Regla: repositorio primero; EM cuando elORM necesita control fino.

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S102

```
interface PedidoRepoCustom { List<Pedido>
  buscarComplejo(Filtro f); }
class PedidoRepoCustomImpl implements PedidoRepoCustom {
  @PersistenceContext EntityManager em; ... } — nombre CLAVE:
  Impl + nombre de la interface.
interface PedidoRepo extends JpaRepository<Pedido,Long>,
  PedidoRepoCustom {} — Spring fusiona ambos.
Para SQL complejo y dinámico sin pelear con JPQL.
```

INTERMEDIO

SPRING BOOT
RESPUESTA

Spring Data JPA

S105

Spring Data JDBC: ORM MÍNIMO — mapea agregados a tablas sin lazy loading, sin dirty checking, sin sesiones; tú escribes SQL simple. Menos sorpresas, más predecible.
Spring Data JPA (Hibernate): relaciones lazy, caché, dirty checking, JPQL — más features, más complejidad.
Regla: dominio simple con agregados planos → Data JDBC; dominio relacional rico → JPA.

INTERMEDIO

SPRING BOOT ★
RESPUESTA

Spring Data JPA

S104

Acceso JDBC SIN el boilerplate de abrir/cerrar Connection, Statement y ResultSet — tú pones el SQL y los parámetros:
jdbc.query("SELECT * FROM users WHERE id = ?", rowMapper, id);
Ideal para: queries puntuales, reportes, batch SQL directo, apps sin JPA.
Maneja recursos y excepciones (traduce SQLException a DataAccessException).
Menos mágico que JPA: tú controlas cada SQL.

INTERMEDIO

SPRING BOOT

Transacciones

S106

@Transactional: ¿cómo funciona y qué trampas tiene?

INTERMEDIO

SPRING BOOT

Transacciones

S107

Niveles de aislamiento de transacciones

INTERMEDIO

SPRING BOOT

Transacciones

S108

Transacciones y tests: @Transactional en tests hace rollback

INTERMEDIO

SPRING BOOT

Transacciones

S109

Las 7 propagaciones de @Transactional (cuáles y cuándo)

AVANZADO

SPRING BOOT

Transacciones

S110

readOnly=true: qué garantiza de verdad

AVANZADO

SPRING BOOT

Transacciones

S111

Timeout y trampas del rollback

AVANZADO

SPRING BOOT

Transacciones

S112

¿Qué pasa si @Transactional está en un método private o se llama desde el mismo bean?

AVANZADO

SPRING BOOT

Transacciones

S113

TransactionTemplate: la alternativa programática

AVANZADO

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

SPRING BOOT

Transacciones

S107

RESPUESTA

READ_UNCOMMITTED: lee datos sin confirmar (dirty read).
 READ_COMMITTED: solo datos confirmados (default Postgres/Oracle).
 REPEATABLE_READ: lecturas repetibles (default MySQL/InnoDB).
 SERIALIZABLE: como ejecución en serie.
 Más aislamiento → menos anomalías, más bloqueos.

INTERMEDIO

SPRING BOOT

Transacciones

S106

RESPUESTA

AOP crea un proxy que abre/commit/rollback la transacción.
 Rollback SOLO con RuntimeException por defecto (usa rollbackFor=Exception.class).
 NO funciona: self-invocation (this.metodo()), métodos private/static.
 Propagación: REQUIRED (une tx existente), REQUIRES_NEW (suspende y crea nueva), NESTED.

INTERMEDIO

SPRING BOOT



Transacciones

S109

RESPUESTA

REQUIRED (default): une o crea.
 REQUIRES_NEW: suspende la actual y abre una NUEVA (logs de auditoría que deben persistir aunque falle el negocio).
 NESTED: savepoint dentro de la actual (rollback parcial posible).
 SUPPORTS: usa si existe.
 NOT_SUPPORTED: suspende (código sin tx dentro de una).
 MANDATORY: exige tx existente (servicio interno).
 NEVER: prohíbe.

AVANZADO

SPRING BOOT

Transacciones

S108

RESPUESTA

@Transactional en un test: cada test corre en tx que se revierte al final — tests aislados y rápidos SIN limpiar BD.
 Trampas: no ves el commit real (problemas de constraints deferred, triggers de commit no se prueban), y con Testcontainers/threads separados NO comparten la tx (usa commit real ahí o TestTransaction).
 TestTransaction framework para control manual.

INTERMEDIO

SPRING BOOT

Transacciones

S111

RESPUESTA

@Transactional(timeout=30): rollback si excede (protege locks).
 Rollback por defecto SOLO en RuntimeException/Error; checked exceptions NO revierten → rollbackFor = Exception.class si el negocio lanza checked.
 noRollbackFor para excepciones 'esperables' que no deben revertir el trabajo ya hecho.

AVANZADO

SPRING BOOT

Transacciones

S110

RESPUESTA

Hint al driver y al ORM: Hibernate desactiva dirty checking y flush → consultas mucho más rápidas en lecturas masivas.
 En BD con soporte (Postgres SET TRANSACTION READ ONLY) puede impedir escrituras.
 NO es una garantía de integridad por sí sola: úsalo en métodos de consulta como optimización + intención.

AVANZADO

SPRING BOOT

Transacciones

S113

RESPUESTA

transactionTemplate.execute(status -> { ...; return resultado; });
 Rollback: status.setRollbackOnly() o excepción.
 Ventajas: control explícito, transacciones PARCIALES dentro de un método (sin mover código a otro bean), imposible olvidar el proxy.
 Inyecta PlatformTransactionManager.

AVANZADO

SPRING BOOT

Transacciones

S112

RESPUESTA

NO hace nada (silenciosamente sin transacción):
 1) private/static/final: el proxy CGLIB no puede sobrescribirlo.
 2) Self-invocation: this.metodo() llama directo, SIN pasar por el proxy.
 Fix: mover a otro bean, inyectarse a sí mismo (@Lazy), o TransactionTemplate programático.
 Causa #1 de 'mi transacción no funciona'.

AVANZADO

SPRING BOOT

Transacciones

S114

@Transactional y eventos: post-commit

AVANZADO

SPRING BOOT

Transacciones

S115

¿Qué es el problema de dual-write y cómo se evita?

AVANZADO

SPRING BOOT

Transacciones

S116

Isolation level: anomalías concretas que resuelve cada nivel

AVANZADO

SPRING BOOT

Transacciones

S117

Locks de BD y transacciones largas: por qué son veneno

AVANZADO

SPRING BOOT

AOP

S118

Vocabulario AOP: aspect, join point, pointcut, advice, weaving

AVANZADO

SPRING BOOT

AOP

S119

Tipos de advice y cuál usar

AVANZADO

SPRING BOOT

AOP

S120

Pointcut expressions: los patrones que debes saber

AVANZADO

SPRING BOOT

AOP

S121

@Around con timer: el ejemplo que todo examen pide

AVANZADO

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

SPRING BOOT

Transacciones

S115

RESPUESTA

Escribir en BD + publicar en cola/mensaje son 2 sistemas: puede confirmar uno y fallar el otro → inconsistencia.
Soluciones: Outbox pattern (evento en tabla outbox en la MISMA tx + proceso que publica), eventos en la misma tx a un broker transaccional, o idempotencia + reconciliación.
Base de la consistencia entre servicios.

AVANZADO

SPRING BOOT

Transacciones

S114

RESPUESTA

@TransactionalEventListener(phase = AFTER_COMMIT): el listener corre SOLO si la tx confirmó — para notificaciones, emails, invalidar caché, publicar a otro sistema.
BEFORE_COMMIT, AFTER_ROLLBACK, AFTER_COMPLETION.
@EventListener normal corre DENTRO de la misma tx (verá datos sin confirmar).
Clave para no enviar emails de pedidos que luego fallaron.

AVANZADO

SPRING BOOT

Transacciones

S117

RESPUESTA

Tx larga = locks sostenidos → otros hilos esperan (timeouts en cascada), conexiones del pool ocupadas, replays de GC del COMMIT...
Buenas prácticas: transacciones cortas (abrir dentro del servicio justo para persistir), NUNCA I/O externa (HTTP, email) dentro de la tx, procesar en chunks.
Síntoma: 'connection is not available, request timed out' (HikariCP).

AVANZADO

SPRING BOOT

Transacciones

S116

RESPUESTA

Dirty read (leer no confirmado): evita READ_COMMITTED+.
Non-repeatable read (misma fila cambia entre lecturas): evita REPEATABLE_READ+.
Phantom read (aparecen filas nuevas en el rango): evita SERIALIZABLE (aprox; Postgres usa MVCC).
Default típico: Postgres/Oracle READ_COMMITTED, MySQL REPEATABLE_READ.

AVANZADO

SPRING BOOT

AOP

S119

RESPUESTA

@Before: antes (validación, log de entrada).
@AfterReturning: tras retorno normal.
@AfterThrowing: tras excepción.
@After: tras ambos (finally).
@Around: ENVUELVE todo — puedes decidir si llamar proceed(), medir tiempo, cachear, transformar resultado. Es el más poderoso (timers, rate limiting).

AVANZADO

SPRING BOOT

AOP

S118

RESPUESTA

Aspect: la clase modular que agrupa funcionalidad transversal (@Aspect + @Component).
Join point: punto del programa donde PUEDE intervenir (en Spring AOP: ejecución de métodos).
Pointcut: la EXPRESIÓN que selecciona join points (execution(...)).
Advice: la acción (@Before/@After/@Around).
Weaving: tejer aspects en los beans (Spring: proxies en runtime).

AVANZADO

SPRING BOOT

AOP

S121

RESPUESTA

```
@Around("@annotation(Monitoreado)")
public Object medir(ProceedingJoinPoint pjp) throws Throwable {
    long t0 = System.nanoTime();
    try { return pjp.proceed(); } finally {
        log.info("{} tardó {} ms", pjp.getSignature(),
            (System.nanoTime()-t0)/1_000_000);
    }
}
```

proceed(args) permite MODIFICAR argumentos/resultados. Sin proceed, el método original NO se ejecuta.

AVANZADO

SPRING BOOT

AOP

S120

RESPUESTA

execution(* com.app.service..*(..)) — todo el paquete service y subpaquetes.
execution(public * *(..)), execution(* Servicio*.metodo(..)).
@annotation(com.app.MiAnotacion) — métodos anotados.
within(com.app.*), this/target(bean type), bean(miBean*).
Combinar con &&, ||, !; reutilizar con @Pointcut nombrado.

AVANZADO

SPRING BOOT

AOP

S122

Limitaciones de Spring AOP (proxy-based)

AVANZADO

SPRING BOOT

AOP

S123

Ordenar aspects y handlers con @Order

AVANZADO

SPRING BOOT

AOP

S124

Casos de uso legítimos de AOP en apps de negocio

AVANZADO

SPRING BOOT

AOP

S125

¿AspectJ vs Spring AOP?

AVANZADO

SPRING BOOT

Seguridad

S126

Authentication vs Authorization (con vocabulario Spring)

INTERMEDIO

SPRING BOOT

★

Seguridad

S127

PasswordEncoder: BCrypt y mejores prácticas

INTERMEDIO

SPRING BOOT

★

Seguridad

S128

JWT flow completo en un backend Spring Boot

INTERMEDIO

SPRING BOOT

Seguridad

S129

HS256 vs RS256 para firmar JWT

INTERMEDIO

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

SPRING BOOT

AOP

S123

RESPUESTA

@Order(n): menor = más EXTERNO (corre antes en entry, después en exit).
 Casos: un aspect de retry fuera del de logging; transacción típicamente muy interna.
 También @Order en interceptores, filtros y ExceptionHandlerAdvices.
 Sin order = indefinido entre aspects (no entre categorías distintas).

AVANZADO

SPRING BOOT

AOP

S122

RESPUESTA

Solo métodos públicos visibles por el proxy: private/final/static NO se interceptan.
 Self-invocation (this.metodo()) salta el proxy →
 @Transactional/@Cacheable/@Async 'no funcionan' dentro del mismo bean.
 Solo ejecución de métodos (no campos ni constructores, eso es AspectJ completo).
 Objetos internos creados con new nunca son proxies.

AVANZADO

SPRING BOOT

AOP

S125

RESPUESTA

Spring AOP: proxies dinámicos, solo métodos de beans Spring, runtime weaving — cubre 95% de casos (usar con @EnableAspectJAutoProxy).
 AspectJ completo: bytecode weaving en compilación/carga — intercepta constructores, campos, métodos private/static, objetos no-Spring; requiere procesador/loader (ajc, LTW).
 Usar AspectJ solo si el límite del proxy realmente duele.

AVANZADO

SPRING BOOT

AOP

S124

RESPUESTA

Logging/auditoría estructurada (@Loggable custom), métricas/timing, manejo de errores transversal, rate limiting, autorización custom (@RequirePermission), retry declarativo, enmascarar datos sensibles en logs, propagar tenant/ctx. NO para lógica de negocio (invisible al leer el flujo — 'magia').
 Regla: si lo borras, la app sigue CORRECTA (solo menos observable/protegida).

AVANZADO

SPRING BOOT



Seguridad

S127

RESPUESTA

SIEMPRE almacenar hashes, jamás texto plano.
 DelegatingPasswordEncoder: PasswordEncoderFactories.createDelegatingPasswordEncoder() — soporta {bcrypt}..., migración de formatos.
 BCrypt incluye salt y factor de trabajo (coste 10 default; 10-12 típico).
 Nunca MD5/SHA-1 solos para passwords.

INTERMEDIO

SPRING BOOT

Seguridad

S126

RESPUESTA

Authentication: ¿quién eres? — AuthenticationManager → ProviderManager → AuthenticationProviders (uno valida usuario/pass, otro JWT...).
 Authorization: ¿qué puedes? — AuthorizationManager contra GrantedAuthorities.
 En el contexto:
 SecurityContextHolder.getContext().getAuthentication() (principal, authorities, credentials).

INTERMEDIO

SPRING BOOT

Seguridad

S129

RESPUESTA

HS256: secreto compartido — simple, pero TODO servicio que valida necesita el secreto (riesgo, no se puede verificar 'solo').
 RS256: par privado/público — firmas con clave privada, cualquiera valida con la pública → ideal multi-servicio y OAuth servers.
 Rotación: publicar claves en JWKS endpoint (/well-known/jwks.json) — el resource server las descarga y cachea.

INTERMEDIO

SPRING BOOT



Seguridad

S128

RESPUESTA

1) POST /login con credenciales → AuthenticationManager valida → generas JWT (jjwt/nimbus) con claims (sub, roles, exp) firmado (HS256 secreto / RS256 par de claves).
 2) Cliente envía Authorization: Bearer <token>.
 3) Resource server: oauth2ResourceServer().jwt() valida firma y exp → authorities desde claims (JwtAuthenticationConverter).
 Stateless: sin sesión en servidor.

INTERMEDIO

SPRING BOOT

Seguridad

S130

Validar JWT a mano: qué DEBES chequear

INTERMEDIO

SPRING BOOT

Seguridad

S131

Session vs Stateless: sessionCreationPolicy

INTERMEDIO

SPRING BOOT

★

Seguridad

S132

CSRF: qué es y cuándo aplica en APIs

INTERMEDIO

SPRING BOOT

Seguridad

S133

401 vs 403 en Spring Security y cómo customizar la respuesta

INTERMEDIO

SPRING BOOT

Seguridad

S134

OAuth2 Client: login social/terceros con Spring

INTERMEDIO

SPRING BOOT

Seguridad

S135

OAuth2 Resource Server: proteger tu API con tokens de terceros

INTERMEDIO

SPRING BOOT

Seguridad

S136

Arquitectura de Spring Security: la cadena de filtros

AVANZADO

SPRING BOOT

★

Seguridad

S137

Config moderna: SecurityFilterChain bean (Spring Security 6)

AVANZADO

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

SPRING BOOT

Seguridad

S131

RESPUESTA

IF_REQUIRED (default): crea sesión (apps web clásicas).
 STATELESS: nunca crea ni usa sesión — cada request trae credenciales (JWT). OBLIGATORIO en APIs con tokens.
 NEVER, ALWAYS, INVALIDATE_ON_AUTHENTICATION... para casos mixtos.
 Con STATELESS, CSRF se deshabilita (no hay cookie de sesión que abuse — ver CSRF).

INTERMEDIO

SPRING BOOT

Seguridad

S130

RESPUESTA

1) Firma contra JWKS/secreto.
 2) exp (expiración) y nbf/iat razonables.
 3) iss (emisor esperado) y aud (audiencia = tu servicio).
 4) Algoritmo esperado (rechazar 'alg: none' o cambios).
 Nunca decodificar SIN validar (Base64 NO es seguridad).
 Spring: oauth2ResourceServer().jwt() con issuer-uri o jwk-set-uri hace todo esto.

INTERMEDIO

SPRING BOOT

Seguridad

S133

RESPUESTA

401 no autenticado (sin/vencido token) —
 AuthenticationEntryPoint.
 403 autenticado pero sin permiso — AccessDeniedHandler.
 Customizar: exceptionHandling(e -> e.authenticationEntryPoint(mi401).accessDeniedHandler(mi403)) — devolver JSON
 ProblemDetail en APIs (default redirige a /error o login).

INTERMEDIO

SPRING BOOT



Seguridad

S132

RESPUESTA

Ataque: un sitio malicioso hace el navegador enviar requests con TU cookie de sesión (form oculto, fetch same-site). Solo aplica con autenticación por COOKIE.
 Defensa: token CSRF (Spring: CookieCsrfTokenRepository con HttpOnly=false para SPAs) o SameSite=Lax cookies.
 APIs stateless con Authorization header →
 http.csrf(AbstractHttpConfigurer::disable) es correcto.

INTERMEDIO

SPRING BOOT

Seguridad

S135

RESPUESTA

oauth2ResourceServer().jwt().issuer-uri(https://authserver) —
 descarga JWKS y valida.
 Roles desde claims: JwtAuthenticationConverter →
 setJwtGrantedAuthoritiesConverter (mapear claim 'roles' →
 SimpleGrantedAuthority).
 Props: spring.security.oauth2.resourceserver.jwt.issuer-uri.
 El auth server (Keycloak, Auth0, Identity Platform) emite; tú solo validas.

INTERMEDIO

SPRING BOOT

Seguridad

S134

RESPUESTA

spring-boot-starter-oauth2-client + properties del provider
 (client-id, secret, authorization-grant-type, redirect-uri).
 http.oauth2Login() → flujo Authorization Code completo
 gestionado por Spring.
 Custom: provider propio en
 spring.security.oauth2.client.registration.
 El principal queda OAuth2User (attributes del userinfo).

INTERMEDIO

SPRING BOOT



Seguridad

S137

RESPUESTA

```
@Bean SecurityFilterChain chain(HttpSecurity http) {
    http.csrf(...).authorizeHttpRequests(a -> a.requestMatchers("/public/**").permitAll().anyRequest().authenticated()).oauth2ResourceServer(...).sessionManagement(s ->
        s.sessionCreationPolicy(STATELESS)); return http.build(); }
WebSecurityConfigurerAdapter está DEPRECADO — todo con beans.
```

AVANZADO

SPRING BOOT

Seguridad

S136

RESPUESTA

SecurityFilterChain: una secuencia de filtros servlet antes de tu controller — SecurityContextPersistenceFilter (carga sesión/token), autenticación (UsernamePassword, BearerToken...), ExceptionTranslationFilter (401/403), FilterSecurityInterceptor/AuthorizationFilter (permisos).
 El resultado: SecurityContext con Authentication disponible en cualquier punto.

AVANZADO

SPRING BOOT

Seguridad

S138

UserDetailsService y UserDetails:
implementación típica

AVANZADO

SPRING BOOT

Seguridad

S139

Roles vs Authorities: anotaciones y expresiones

AVANZADO

SPRING BOOT

Seguridad

S140

Multi-tenancy ligero: tenant por header

AVANZADO

SPRING BOOT

Seguridad

S141

¿Cómo probar endpoints protegidos en tests?

AVANZADO

SPRING BOOT

Seguridad

S142

Secrets en Spring Boot: dónde NO ponerlos y dónde sí

AVANZADO

SPRING BOOT

Testing

S143

La pirámide de testing aplicada a Spring Boot

INTERMEDIO

SPRING BOOT ★

Testing

S144

@WebMvcTest: qué levanta y qué debes mockear

INTERMEDIO

SPRING BOOT ★

Testing

S145

MockMvc: el recetario de asserts

INTERMEDIO

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

SPRING BOOT

Seguridad

S139

RESPUESTA

Authority = permiso fino ("pedido:leer"); ROLE_ es una convención de prefijo (ROLE_ADMIN).
 http.authorizeHttpRequests: hasAuthority("x"), hasRole("ADMIN") (añade ROLE_ automáticamente).
 Método: @PreAuthorize("hasRole('ADMIN') or #id == authentication.principal.id") — SpEL con parámetros (#id).
 Habilitar: @EnableMethodSecurity.

AVANZADO

SPRING BOOT

Seguridad

S138

RESPUESTA

UserDetailsService.loadUserByUsername(String) → UserDetails (username, password hash, authorities, enabled).
 Impl con tu repo de usuarios; Spring valida password con PasswordEncoder.matches.
 Si usas JWT/OAuth sin usuarios internos, quizá no necesites UserDetailsService (custom provider o JWT decoder).

AVANZADO

SPRING BOOT

Seguridad

S141

RESPUESTA

@WithMockUser(username, roles={"ADMIN"}) sobre el test o método.
 @WithJwt / @WithSecurityContext para JWTs custom.
 MockMvc + SecurityMockMvcRequestPostProcessors.jwt() (de spring-security-test): .with(jwt().authorities(...)).
 CSRF en MockMvc: .with(csrf()) para POST.

AVANZADO

SPRING BOOT

Seguridad

S140

RESPUESTA

Filter propio al inicio de la chain: extrae X-Tenant-Id → guarda en ThreadLocal/TenantContext.
 Datasource o schema dinámico: AbstractRoutingDataSource decide por tenant.
 Validar SIEMPRE que el tenant coincida con los claims del token (no confiar en el header solo).
 Limpieza del ThreadLocal al final (pools de hilos).

AVANZADO

SPRING BOOT

Testing

S143

RESPUESTA

Base (muchos): tests UNITARIOS de lógica pura sin Spring (rápidos, JUnit+Mockito).
 Medio: tests de SLICE — @WebMvcTest, @DataJpaTest, @JsonTest (levantan SOLO esa capa, segundos).
 Cima (pocos): @SpringBootTest integración completa (+Testcontainers con BD real).
 Anti-patrón: todo @SpringBootTest → suite de horas.

INTERMEDIO

SPRING BOOT

Seguridad

S142

RESPUESTA

NO: application.yml commiteado, código, images, logs.
 SÍ: variables de entorno (12-factor), Secret Manager/Vault (leer al arranque o con Spring Cloud Vault), archivos montados por K8s.
 Placeholder: \${DB_PASS} desde env.
 Rotación: backend con relectura o restart controlado; jamás loguear properties con password (logging.mask).

AVANZADO

SPRING BOOT ★

Testing

S145

RESPUESTA

mockMvc.perform(get("/api/pedidos/1").header(...)).andExpect(status().isOk()).andExpect(jsonPath("\$.cliente.nombre").value("Ana")).andExpect(jsonPath("\$.items", hasSize(3))).andDo(print()) / document().
 POST con body: contentType(APPLICATION_JSON).content(objectMapper.writeValueAsString(dto)).
 jsonPath usa expresiones JsonPath (\$) = raíz.

INTERMEDIO

SPRING BOOT ★

Testing

S144

RESPUESTA

Levanta SOLO la capa web: controllers, @ControllerAdvice, filtros/converters MVC, MockMvc autoconfigurado.
 NO levanta services/repos → inyectas @MockitoBean (Spring Boot 3.4+, antes @MockBean) MiService.
 Uso: validar mappings, status codes, validación de DTOs, manejo de errores, serialización.

INTERMEDIO

SPRING BOOT

Testing

S146

@DataJpaTest y repositorios: qué prueba y qué trae

INTERMEDIO

SPRING BOOT

Testing

S147

Testcontainers: BD real en Docker para tus tests

INTERMEDIO

SPRING BOOT

Testing

S148

@MockitoBean (ex @MockBean) vs Mockito @Mock: diferencia clave

INTERMEDIO

SPRING BOOT

Testing

S149

Mockito: verify, ArgumentCaptor y stubbing útil

INTERMEDIO

SPRING BOOT

Testing

S150

@SpringBootTest con puerto real: @LocalServerPort y TestRestTemplate

INTERMEDIO

SPRING BOOT

Testing

S151

Cómo probar código async y schedulers

INTERMEDIO

SPRING BOOT

Testing

S152

Fixture de datos de prueba: @Sql, builders y fixtures

INTERMEDIO

SPRING BOOT

Testing

S153

AssertJ: los asserts fluidos que reemplazan a JUnit asserts

INTERMEDIO

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

SPRING BOOT

Testing

S147

RESPUESTA

@Testcontainers + @Container static PostgreSQLContainer
 postgres = new PostgreSQLContainer("postgres:16").
 @DynamicPropertySource registra
 spring.datasource.url/username/password dinámicos.
 Beneficio: el MISMO motor que producción, migraciones Flyway
 reales ejecutadas.
 Costo: Docker requerido y segundos extra (contenedores
 estáticos reutilizados lo mitigan).

INTERMEDIO

SPRING BOOT

Testing

S146

RESPUESTA

Levanta JPA + DataSource EM_MEMORIA (H2 por defecto) +
 repositorios; transacción por test con rollback.
 Configurar BD real: @AutoConfigureTestDatabase(replace =
 NONE) + Testcontainers.
 Probar: queries derivadas/@Query críticas, constraints, mapeo de
 entidades, paginación.
 H2 ≠ tu motor: para SQL nativo complejo usa Testcontainers con
 el motor real.

INTERMEDIO

SPRING BOOT

Testing

S149

RESPUESTA

when(repo.findById(1L)).thenReturn(Optional.of(entidad));
 thenThrow para excepciones.
 verify(repo).save(captor.capture()); captor.getValue() → assert del
 objeto guardado.
 never(), times(2), any(), eq() — mezclar matchers requiere TODOS
 matchers.
 anyMatch: when(x.metodo(anyString())).thenReturn(...).
 Comportamiento default: null/0/false (relaxed).

INTERMEDIO

SPRING BOOT

Testing

S148

RESPUESTA

@MockitoBean: Spring reemplaza el bean EN EL CONTEXTO —
 para tests con Spring que inyectan dependencias.
 @Mock (Mockito): mock puro sin Spring — para tests unitarios
 con constructor (new Servicio(mock)).
 Trampa: @MockitoBean crea un NUEVO contexto (cache de
 contextos se rompe si varían) → suite lenta; minimizar
 combinaciones distintas.

INTERMEDIO

SPRING BOOT

Testing

S151

RESPUESTA

Async: capturar el CompletableFuture y .get(timeout), o Awaitility:
 await().atMost(5, SECONDS).untilAsserted() -> assertThat(...).
 Scheduled: no duermas 5 min — extraer la lógica a un método
 testeable y llamarlo directo; o @TestPropertySource con cron
 acelerado.
 Eventos: @RecordApplicationEvents + events.stream(...).

INTERMEDIO

SPRING BOOT

Testing

S150

RESPUESTA

@SpringBootTest(webEnvironment = RANDOM_PORT): levanta el
 servidor completo en un puerto libre.
 @LocalServerPort int port inyecta el puerto; cliente:
 TestRestTemplate (auto-configurado) o RestClient apuntando a
 localhost:port.
 Para pruebas end-to-end del flujo HTTP real (auth incluida) —
 pocas, críticas.

INTERMEDIO

SPRING BOOT

Testing

S153

RESPUESTA

assertThat(lista).hasSize(3).extracting("nombre").containsExactly("a", "b").
 assertThat(excepcion).assertInstanceOf(...).hasMessageContaining(...).
 assertThat(dto).usingRecursiveComparison().ignoringFields("id").isEqualTo(otro).
 withFailMessage, satisfies(v -> ...). Viene incluido en starter-test.

INTERMEDIO

SPRING BOOT

Testing

S152

RESPUESTA

@Sql("/data/test-usuarios.sql") ejecuta scripts antes del test
 (@Sql(executionPhase = AFTER_TEST_METHOD) para limpiar).
 @BeforeEach con builders de entidades (ObjectMother/builder
 pattern) — datos legibles en el propio test.
 Base de datos en memoria por test: @Transactional rollback
 automático.
 Evitar shared mutable state entre tests (orden aleatorio ayuda:
 JUnit method order RANDOM).

INTERMEDIO

SPRING BOOT

Testing

S154

¿Cómo medir y actuar sobre cobertura (JaCoCo)?

INTERMEDIO

SPRING BOOT

Testing

S155

Test de una clase con @Transactional: ¿se prueba la transacción?

INTERMEDIO

SPRING BOOT

★

Producción y observabilidad

S156

¿Qué ofrece Spring Boot Actuator?

BASICO

SPRING BOOT

Producción y observabilidad

S157

Micrometer: cómo Spring Boot mide todo

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S158

Health indicator custom y health groups

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S159

Logging con traceld correlacionado (MDC + Micrometer Tracing)

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S160

Config de logging que debe saber un backend

INTERMEDIO

SPRING BOOT

★

Producción y observabilidad

S161

Graceful shutdown y readiness durante deploy

INTERMEDIO

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

SPRING BOOT

Testing

S155

RESPUESTA

El test @Transactional hace rollback → el commit real NO se ejecuta: constraints deferred, triggers ON COMMIT y behaviours de aislamiento no se validan.

Para validar eso: test SIN @Transactional + limpieza manual, o TestTransaction.commit() explícito dentro del test. Elije conscientemente.

INTERMEDIO

SPRING BOOT

Testing

S154

RESPUESTA

Plugin jacoco-maven-plugin: report en target/site/jacoco; check con reglas (BUNDLE line ratio 0.7) que fallen el build si bajo. Realismo: 100% es ruido; apunta a lógica de negocio alta, config/trivial baja.

Cobertura no prueba decisiones: un test que ejecuta sin asserts cubre igual (mutation testing lo delata — PIT).

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S157

RESPUESTA

Micrometer es la fachada de métricas (SLF4J para métricas): Timer, Counter, Gauge, DistributionSummary. registry.timer("pedidos.proceso", "tipo", "web").record(...); o @Timed/@Counted.

/exporter: /actuator/prometheus → Prometheus → Grafana. Métricas auto: JVM, HTTP (duración, status), Hikari, cachés.

INTERMEDIO

SPRING BOOT



Producción y observabilidad

S156

RESPUESTA

Endpoints de operación:

/actuator/health (con liveness/readiness), /metrics (JVM, HTTP, pools), /info, /env, /prometheus (exporter para Grafana).

Se exponen con management.endpoints.web.exposure.include. Base del monitoreo y de los probes de K8s.

BASICO

SPRING BOOT

Producción y observabilidad

S159

RESPUESTA

Micrometer Tracing (ex Sleuth) propaga traceId/spanId en MDC → patrón de log: %X{traceId}.

logging.pattern.level="\${LOG_LEVEL} [\${traceId:-}]".

Un request a través de 5 servicios = un traceId en los logs de todos → grep único para investigar.

W3C traceparent o B3 headers entre servicios.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S158

RESPUESTA

implements HealthIndicator: Health.up().withDetail("latencia", ms).build() / down(excepción).

Bean nombrado xxxHealthIndicator → /actuator/health/{xxx}.

Health groups:

management.endpoint.health.group.readiness.include=db,ping → /actuator/health/readiness para K8s.

Cuidado: dependencia lenta DOWN = app fuera del balanceo (usar cache/status opcional).

INTERMEDIO

SPRING BOOT



Producción y observabilidad

S161

RESPUESTA

server.shutdown=graceful +

spring.lifecycle.timeout-per-shutdown-phase=30s: al recibir SIGTERM, la app deja de aceptar requests NUEVOS, termina los en curso y recién cierra.

Combinar con readiness probe: quitar del balanceo ANTES de matar (preStop hook en K8s).

Evita 502s y requests cortados en cada deploy.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S160

RESPUESTA

logging.level.root=INFO, logging.level.com.miapp=DEBUG (por paquete).

Pattern con timestamp, hilo, level, logger, traceId.

Salida estructurada JSON en prod (logstash-logback-encoder) para ingest.

Rolling: size + días retenidos. NUNCA loguear passwords/tokens/PII completa.

INTERMEDIO

SPRING BOOT

Producción y observabilidad

S162

Dockerizar Spring Boot: layered jars

AVANZADO

SPRING BOOT

Producción y observabilidad

S163

Virtual threads en Spring Boot 3.2+: un flag

AVANZADO

SPRING BOOT

Producción y observabilidad

S164

GraalVM native image en Spring Boot: qué cambia

AVANZADO

SPRING BOOT

Producción y observabilidad

S165

Perfil prod: checklist de propiedades

AVANZADO

SPRING BOOT

Async, caché y scheduling

S166

@Async: setup y reglas para que funcione

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S167

@Scheduled: cron, fixedRate vs fixedDelay y zona

INTERMEDIO

SPRING BOOT

★ Async, caché y scheduling

S168

@Cacheable, @CacheEvict, @CachePut: semántica exacta

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S169

Caffeine: configuración típica de caché local

INTERMEDIO

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

SPRING BOOT

Producción y observabilidad

S163

RESPUESTA

spring.threads.virtual.enabled=true → Tomcat, @Async y schedulers usan hilos virtuales.
 I/O-bound masivo con código bloqueante simple: miles de requests concurrentes sin WebFlux ni pools gigantes.
 Cuidado: synchronized prolongado (pinning), CPU-bound no mejora, y pools de conexiones quedan como cuello (ajustar Hikari).

AVANZADO

SPRING BOOT

Producción y observabilidad

S162

RESPUESTA

spring-boot-jarmode-layers: java -Djarmode=tools -jar app.jar
 extract → Dockerfile con COPY de capas separadas (dependencies, snapshot, app) → el cache de Docker solo re-descarga la capa 'app' cuando cambia TU código (imagen rebuild 10x más rápida).
 Buildpacks (bootBuildImage) es la alternativa cero-Dockerfile.

AVANZADO

SPRING BOOT

Producción y observabilidad

S165

RESPUESTA

ddl-auto=validate, show-sql=false, logging INFO + JSON,
 actuator expone solo health/info/prometheus (nada de /env),
 CORS restricto, TLS en el LB, secrets por env/Secret Manager,
 timeouts de clientes configurados, Hikari sized, graceful
 shutdown, management probes habilitados
 (management.endpoint.health.probes.enabled=true).

AVANZADO

SPRING BOOT

Producción y observabilidad

S164

RESPUESTA

AOT: compila a binario nativo — arranque <100ms y footprint mínimo (ideal serverless/scale-to-zero).
 Costos: build lento, reflexión/JPA dinámicos requieren hints, alguna librería incompatible.
 Spring Boot 3 trae soporte AOT + plugins. Evaluar solo si el arranque/frío importa de verdad.

AVANZADO

SPRING BOOT

Async, caché y scheduling

S167

RESPUESTA

@Scheduled(cron = "0 0 2 * * *", zone = "America/Lima") — 2 a.m. (seg min hora día mes díaSem).
 fixedRate = cada N ms DESDE EL INICIO; fixedDelay = N ms DESDE EL FIN (nunca solapa) — el seguro para tareas lentas.
 Requiere @EnableScheduling. default es single-thread: tareas largas retrasan las demás → configurar TaskScheduler pool.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S166

RESPUESTA

1) @EnableAsync en una @Configuration.
 2) Configurar executor: AsyncConfigurer o spring.task.execution.* (default SimpleAsyncTaskExecutor crea un hilo por llamada — ¡peligroso en prod!).
 3) Método público llamado DESDE OTRO bean (proxy).
 Retorno: void o Future/CompletableFuture. Excepciones: AsyncUncaughtExceptionHandler si void.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S169

RESPUESTA

Caffeine.newBuilder().maximumSize(10_000).expireAfterWrite(Duration.ofMinutes(10)).refreshAfterWrite(5min).
 .maximumWeight + weigher para limitar por memoria.
 recordStats() para métricas (hit rate).
 En Spring: SpringApplicationBuilder o CaffeineCacheManager;
 expiración variable con expireAfter(Expiry custom).

INTERMEDIO

SPRING BOOT ★

Async, caché y scheduling

S168

RESPUESTA

@Cacheable("productos", key="#id"): si está en caché NO ejecuta el método.
 @CachePut: SIEMPRE ejecuta y actualiza la caché (para writes que deben refrescar).
 @CacheEvict: invalida (allEntries=true borra todo).
 unless="#result == null" evita cachear vacíos; condition filtra.
 Requiere @EnableCaching y un CacheManager (Caffeine local, Redis distribuido).

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S170

Caché con Redis en Spring: consideraciones

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S171

Cuándo cachear y cuándo NO (decisiones que evaluán en entrevista)

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S172

Eventos de aplicación: @EventListener y publishing

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S173

Schedulers en cluster: por qué necesitas un lock (ShedLock)

AVANZADO

SPRING BOOT

Async, caché y scheduling

S174

Cache stampede (thundering herd): qué es y cómo mitigarlo

AVANZADO

SPRING BOOT

Async, caché y scheduling

S175

Cuándo @EventListener vs @TransactionalEventListener

AVANZADO

SPRING BOOT

★ Spring Cloud y resiliencia

S176

¿Qué es un Circuit Breaker (Resilience4j)?

INTERMEDIO

SPRING BOOT

★ Spring Cloud y resiliencia

S177

Spring Cloud Gateway: qué hace y config básica

INTERMEDIO

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

SPRING BOOT

Async, caché y scheduling

S171

RESPUESTA

SÍ: lecturas frecuentes de datos poco cambiantes (catálogos, configs, cálculos costosos), alivio a BD bajo picos.
 NO: datos que cambian con cada write, respuestas personalizadas sin clave correcta, datos críticos sin tolerancia a stale, tasa de hit baja (overhead gratis).
 TTL + invalidación + stampede protection (refreshAhead/singleflight) = el kit completo.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S170

RESPUESTA

spring-boot-starter-data-redis + RedisCacheManager (Jackson2JsonRedisSerializer para guardar DTOs).
 Claves: nombres de caché como prefijo (productos:42).
 TTL obligatorio (todo dato cached expira), invalidación cruzada entre servicios por eventos.
 Serialización estable: sin classnames frágiles, versionar DTOs cached.

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S173

RESPUESTA

Con 3 réplicas, @Scheduled corre 3 VECES (una por instancia).
 Soluciones: ShedLock (@SchedulerLock(name, lockAtMostFor) + tabla/Redis: solo una instancia ejecuta), Quartz clusterizado, o mover al scheduler del cloud (Cloud Scheduler → endpoint, Cloud Tasks).
 Elegir idempotencia SIEMPRE como red de seguridad.

AVANZADO

SPRING BOOT

Async, caché y scheduling

S172

RESPUESTA

publisher.publishEvent(new PedidoCreadoEvent(id)) → @EventListener corren sincrónicos EN EL MISMO HILO/tx por defecto.
 Async: @Async sobre el listener.
 Order con @Order. Eventos desacoplan módulos internos (no son mensajes de microservicios).
 Payload inmutable; listener NO debe fallar el flujo principal (try/catch o async).

INTERMEDIO

SPRING BOOT

Async, caché y scheduling

S175

RESPUESTA

@EventListener: inmediato (misma tx — ve cambios sin confirmar; si el listener escribe, comparte tx).
 @TransactionalEventListener(AFTER_COMMIT): solo si la tx CONFIRMA — el estándar para efectos externos (emails, colas, invalidar caché).
 Trampa AFTER_COMMIT: nueva tx propia para escribir (REQUIRES_NEW o TransactionTemplate).
 fallbackExecution=true si no hay tx.

AVANZADO

SPRING BOOT

Async, caché y scheduling

S174

RESPUESTA

Clave expira → 1000 requests concurrentes golpean la BD simultáneamente para rehacerla.
 Mitigación: lock/singleflight (uno recalcula, el resto espera el nuevo valor), refreshAhead (renovar antes de expirar), TTL con jitter aleatorio, y fallback a valor stale mientras se recalcula (serve-stale).

AVANZADO

SPRING BOOT



Spring Cloud y resiliencia

S177

RESPUESTA

Gateway reactivo (Netty) como entrada única: rutas por path/host con filtros.
 spring.cloud.gateway.routes: id, uri, predicates (Path=/api/**), filters (StripPrefix=1, AddRequestHeader, Retry, CircuitBreaker).
 Filters custom: GatewayFilterFactory — auth, rate limit (RequestRateLimiter con Redis).
 No pongas lógica de negocio en el gateway.

INTERMEDIO

SPRING BOOT



Spring Cloud y resiliencia

S176

RESPUESTA

Patrón anti-cascada: si un servicio remoto falla N veces, el circuito se ABRE y falla rápido sin llamarlo (fallback); tras un tiempo pasa a HALF-OPEN y prueba de nuevo; si funciona, se CIERRA.
 @CircuitBreaker(name="servicio", fallbackMethod="respaldo").

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S178

Resilience4j: los 4 módulos y anotaciones

INTERMEDIO

SPRING BOOT

★

Spring Cloud y resiliencia

S179

Retry bien hecho: backoff, jitter y cuándo NO reintentar

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S180

OpenFeign: cliente declarativo y sus settings críticos

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S181

CircuitBreaker config: qué significa cada número

AVANZADO

SPRING BOOT

Spring Cloud y resiliencia

S182

Spring Cloud Config Server: qué aporta hoy

AVANZADO

SPRING BOOT

Troubleshooting clásico

S183

NoUniqueBeanDefinitionException: diagnóstico en 3 pasos

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S184

BeanCurrentlyInCreationException (dependencia circular): salidas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S185

Port 8080 was already in use: soluciones rápidas

INTERMEDIO

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

SPRING BOOT



Spring Cloud y resiliencia

S179

RESPUESTA

maxAttempts=3, waitDuration=200ms, enableExponentialBackoff (x2), enableRandomizedJitter (evita sincronización de reintentos). Reintentar SOLO errores transitorios (503, timeouts, conexiones) — NO 400/401/404 (deterministas). Idempotencia obligatoria en la operación reintentada. Retry + circuit breaker juntos: el breaker evita martillar cuando todo está caído.

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S178

RESPUESTA

@CircuitBreaker (corta tras N fallos), @Retry (n reintentos + backoff), @RateLimiter (límite por periodo), @Bulkhead (semáforo o pool de hilos), @TimeLimiter (timeout para futures). Combinables en un método (orden configurable). fallbackMethod = "miMetodo(Throwable)" con firma compatible. Métricas expuestas vía Micrometer: estado del breaker en /metrics.

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S181

RESPUESTA

slidingWindowSize=10 (ventana de llamadas evaluada), failureRateThreshold=50 (% de fallos para abrir), waitDurationInOpenState=30s (tiempo abierto), permittedNumberOfCallsInHalfOpenState=3, slowCallRateThreshold + slowCallDurationThreshold (latencia alta cuenta como fallo). En yml: resilience4j.circuitbreaker.instances.miServicio.*

AVANZADO

SPRING BOOT

Spring Cloud y resiliencia

S180

RESPUESTA

@FeignClient(name="pagos", url="\${servicios.pagos.url}", configuration=FeignCfg.class) interface PagosClient {
@PostMapping("/cargar") Carga cargar(@RequestBody CargaReq req); }
@EnableFeignClients. Timeouts SIEMPRE (connect/read por client), retry acotado, error decoder (ErrorDecoder → excepciones de negocio), y circuit breaker integrado (circuitbreaker.enabled=true).
Interceptors para tokens: RequestInterceptor con Authorization header.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S183

RESPUESTA

1) Leer el mensaje: lista los beans candidatos.
2) Decidir: @Primary (default), @Qualifier (punto de inyección), o renombrar el campo para que coincida con el nombre del bean.
3) Si es involuntario: revisar componentes duplicados por escaneo (paquete repetido, clases de prueba escaneadas).
En tests: @MockitoBean reemplaza y elimina la ambigüedad.

INTERMEDIO

SPRING BOOT

Spring Cloud y resiliencia

S182

RESPUESTA

Centraliza properties por app/perfil en un repo Git: los servicios las cargan al arranque desde http://config-server/app/prod.
@RefreshScope + actuador/refresh recarga valores SIN redeploy (con bus de eventos: refresca todas las réplicas).
Alternativa moderna en k8s: ConfigMaps + reloder (spring-cloud-kubernetes). Aun así, el patrón externalized config es el que importa.

AVANZADO

SPRING BOOT

Troubleshooting clásico

S185

RESPUESTA

Ver quién ocupa: Windows netstat -ano | findstr :8080 + taskkill /PID x /F (o cambiar server.port=0/otro).
En dev: apagar el otro servicio (o el Docker que publicó el puerto), o correr con --server.port=9090.
En tests: RANDOM_PORT siempre.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S184

RESPUESTA

Mensaje típico: 'Requested bean is currently in creation'.
Opciones en orden de preferencia: 1) Rediseñar — extraer lo común a un tercer bean. 2) @Lazy en UNA de las inyecciones. 3) Setter injection en lugar de constructor (solo como parche).
La circularidad casi siempre indica responsabilidades mal separadas.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S186

Cannot resolve property / Property or field 'x' cannot be found: causas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S187

HikariPool - Connection is not available, request timed out: por qué ocurre

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S188

JSON parse error / 415 Unsupported Media Type: checklist

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S189

Infinite recursion con Jackson (StackOverflow): 4 salidas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S190

Tabla not found / SQLGrammarException al arrancar: causas típicas

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S191

El test pasa en local pero falla en CI: los 5 clásicos

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S192

502/504 detrás de un load balancer con Spring Boot: dónde mirar

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S193

Cómo diagnosticar un endpoint lento (metodología)

INTERMEDIO

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

SPRING BOOT



Troubleshooting clásico

S187

RESPUESTA

Pool agotado: conexiones en uso > maximumPoolSize y nadie libera.

Causas: transacciones largas (I/O externa dentro de tx), locks de BD, conexiones sin cerrar (streams no cerrados), pool chico para la carga.

Fixes: tx cortas, raise pool (con margen de BD), leakDetectionThreshold=30000 para culprits, métricas de Hikari (active/idle/wait).

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S186

RESPUESTA

- 1) En SpEL/JPQL: typo o nombre de propiedad inexistente en la entidad (JPQL usa campos de entidad, NO columnas).
 - 2) En @ConfigurationProperties: relaxed binding mal aplicado (lista sin corchetes).
 - 3) Typo en application.yml bajo el prefijo equivocado.
- Validar: arranque con --debug y CONDITIONS report; correr el query a mano.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S189

RESPUESTA

Relación A→B→A serializando.

- 1) DTOs (la solución real).
- 2) @JsonIgnore en el lado inverso.
- 3) @JsonManagedReference (padre) + @JsonBackReference (hijo).
- 4) @JsonIdentityInfo(generator=ObjectIdGenerators.PropertyGenerator.class, property="id").
- 5) @JsonProperty(access=WRITE_ONLY) para entrada sin salida.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S188

RESPUESTA

- 415: el cliente NO envía Content-Type: application/json, o el endpoint exige consumes distinto.
- JSON error 400: payload mal formado, campo con tipo incorrecto (string vs number), DTO sin constructor vacío/setters (Jackson necesita), o getter/setter faltantes.
- Debug: loguear el body recibido (solo en dev) y probar la serialización del DTO aislada.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S191

RESPUESTA

- 1) Depende de recursos locales (puerto, archivo, red) — mockear/contenerizar.
 - 2) TZ/locáles distintas → fechas rotas: fijar TZ/UTF-8 en el runner.
 - 3) Orden de tests / estado compartido.
 - 4) Docker indisponible para Testcontainers.
 - 5) Flaky por timing (async sin await — usar Awaitility).
- Correr tests en paralelo/aleatorio localmente los vuelve robustos.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S190

RESPUESTA

- 1) ddl-auto sin create + BD vacía y sin migraciones.
 - 2) Flyway con versión de script menor a la registrada (schema history).
 - 3) Naming: tabla snake_case en BD vs entidad sin naming strategy (spring.jpa.hibernate.naming.physical-strategy).
 - 4) Esquema/schema search path distinto (Postgres).
- Validar contra la BD real, no contra la entidad.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S193

RESPUESTA

- 1) Medir por capas: gateway log → micrometer http.server.requests.percentile → logs con timing.
- 2) Activar show-sql/log de Hikari SOLO en repro local; buscar N+1 y queries sin índice (EXPLAIN).
- 3) Revisar llamadas externas (timeouts, latencia del cliente HTTP).
- 4) GC/hilos en JFR si todo interno luce bien.
- 5) Fix + test de carga que demuestre la mejora.

INTERMEDIO

SPRING BOOT

Troubleshooting clásico

S192

RESPUESTA

- 502: la app rechaza/cierra — durante deploy sin graceful shutdown, proceso caído, puerto equivocado en el target group.
- 504: la app tarda más que el timeout del LB/ingress (60s default) — perfil de lentitud, timeouts del LB ajustados o request asincrónico.
- Confirmar: logs de la app a esa hora + health del backend + readiness antes del switch de tráfico.

INTERMEDIO

SPRING BOOT

Ecosistema y build

S194

Maven lifecycle: clean, compile, test, package, verify, install, deploy

BASICO

SPRING BOOT

Ecosistema y build

S195

scope de dependencias en Maven

BASICO

SPRING BOOT

Ecosistema y build

S196

dependencyManagement vs dependencies (y BOMs)

BASICO

SPRING BOOT

Ecosistema y build

S197

Exclusions y conflictos: el flujo de diagnóstico

BASICO

SPRING BOOT

Ecosistema y build

S198

Maven vs Gradle: diferencias prácticas

BASICO

SPRING BOOT

Ecosistema y build

S199

Lombok: anotaciones y las 2 trampas

BASICO

SPRING BOOT

Trampas de entrevista

S200

¿Dos beans del mismo tipo pero uno con @Profile: es ambiguo?

BASICO

SPRING BOOT

WebFlux y reactivo

S201

WebFlux vs MVC: cuál elegir con datos en la mano

INTERMEDIO

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

SPRING BOOT

Ecosistema y build

S195

RESPUESTA

compile (default): todo. provided: compilación sí, runtime NO lo empaca (servlet-api, Lombok). runtime: solo runtime. test: solo tests (JUnit, Mockito). import: solo en dependencyManagement (importar BOMs).

Transitivo: compile fluye; test/provided no.

Conflicto de versiones: gana el más CERCANO al root (nearest wins) — dependency:tree para diagnosticar.

BASICO

SPRING BOOT

Ecosistema y build

S194

RESPUESTA

Fases en orden: compile < test < package (jar/war) < verify (integration tests, jacoco check) < install (al repo local) < deploy (a repo remoto).

Ejecutar 'mvn package' corre TODAS las fases anteriores.

Plugins se enganchan a fases (surefire→test, boot plugin→repackage crea el jar ejecutable).

-mvn clean borra target/.

BASICO

SPRING BOOT

Ecosistema y build

S197

RESPUESTA

mvn dependency:tree -Dincludes=com.google.guava:guava → quién trae qué versión.

Excluir en la dependencia culpable:

<exclusions><exclusion>...</exclusion></exclusions>, y declarar la versión deseada explícita (o en dependencyManagement).

Síntomas típicos: NoSuchMethodError/ClassNotFoundError en runtime por versiones mezcladas.

BASICO

SPRING BOOT

Ecosistema y build

S196

RESPUESTA

dependencyManagement: declara VERSIONES sin incluir la dependencia (los hijos heredan versión).

BOM (bill of materials): pom solo de versiones importado con scope=import — spring-boot-dependencies alinea 300 librerías.

Resultado: cambiar versión en un solo lugar; sin versiones duplicadas ni 'inferno de dependencias'.

BASICO

SPRING BOOT

Ecosistema y build

S199

RESPUESTA

@Getter/@Setter, @RequiredArgsConstructor (inyección de dependencias final — la base del estilo constructor), @Builder, @Data (¡incluye equals/hashCode/toString con TODOS los campos — peligro en entidades JPA!), @Slf4j, @Value (inmutable), @EqualsAndHashCode(exclude=onlyExplicitlyIncluded).

Trampas: IDE sin plugin no compila; @Data en entidades rompe equals/hashCode con lazy/collections.

BASICO

SPRING BOOT

Ecosistema y build

S198

RESPUESTA

Maven: XML declarativo, convención sólida, el más extendido en empresas; build ligeramente más lento.

Gradle: Kotlin/Groovy DSL, build incremental y cacheado (más rápido), flexible (a veces demasiado).

Ambos con mvnw/gradlew wrapper para versionar el build tool.

Elige el estándar del equipo.

BASICO

SPRING BOOT

WebFlux y reactivo

S201

RESPUESTA

MVC (bloqueante, 1 request = 1 hilo): óptimo para CRUD, JPA, ecosistema maduro. Con 200 hilos sostiene cientos de req/s.

WebFlux (no bloqueante, pocos event-loop threads): gana con MUCHA concurrencia + I/O (streaming, proxies, backpressure), no con más CPU.

Regla: si tu stack es JPA/JDBC (bloqueante), WebFlux NO ayuda — jdbc bloquea el event loop. R2DBC para reactivo real.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S200

RESPUESTA

No, si los perfiles son excluyentes: solo un bean EXISTE por perfil activo → sin ambigüedad en runtime.

Pero en tests sin perfil activo ambos pueden no existir o existir según configuración.

Patrón: impl por entorno (FakeNotificador dev / EmailNotificador prod) — ejemplo perfecto de @Profile.

BASICO

SPRING BOOT

WebFlux y reactivo

S202

Mono y Flux: semántica exacta

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S203

¿Puedes inyectar en un campo static? ¿Y por qué no?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S204

¿Cuándo se crea un bean @Lazy si nadie lo usa? ¿Y si otro bean lo inyecta?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S205

¿Un bean prototype con @PostConstruct corre por cada instancia?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S206

¿Qué pasa si dos @Bean devuelven el MISMO tipo y un tercero lo inyecta por constructor?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S207

¿getBean() dentro de un service es buena idea?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S208

¿@Transactional en la clase Y en un método: qué gana? ¿Y anidado en un bean interno?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S209

¿Diferencia entre @ComponentScan con lazy-init global y @Lazy?

INTERMEDIO

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

SPRING BOOT

Trampas de entrevista

S203

RESPUESTA

NO. Spring inyecta en instancias de beans; los static pertenecen a la clase, y el contenedor no los toca (además fomenta estado global no testeable).

Soluciones: ponerlo en un bean normal e inyectar, o pattern holder inicializado por el contexto (ApplicationContextHolder) solo como último recurso legado.

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S202

RESPUESTA

Mono<T>: 0 o 1 elemento (o vacío/error) — la versión reactiva de Optional/Future.

Flux<T>: 0..N elementos con señal de completado y backpressure.

NADA se ejecuta sin SUBSCRIBIRSE: flux.map(...) devuelve otro flujo frío; el trabajo arranca en subscribe/return al framework.

Operaciones declarativas: map/flatMap/filter/concatWith/merge.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S205

RESPUESTA

Sí: el ciclo completo (inyección, @PostConstruct) corre por CADA creación.

Pero @PreDestroy NO se llama: Spring no rastrea prototipos tras entregarlos — cleanup manual por el cliente (destroy callback o try-with-resources).

Por eso prototipos con recursos necesitan destroyMethod o gestión explícita.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S204

RESPUESTA

Nunca se crea si nadie lo pide.

Con @Lazy en la inyección: el CONSUMIDOR recibe un proxy; el bean real se crea al PRIMER método llamado.

Sin @Lazy pero con singleton eager (default): se crea al arrancar aunque nadie lo use.

@Lazy en @Configuration(proxyBeanMethods) es otra cosa: proxies de métodos @Bean.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S207

RESPUESTA

Casi nunca: service locator pattern oculta dependencias (constructor no muestra lo que usa), imposible de mockear limpio, y ancla al contexto.

Excepciones aceptables: código legacy no-manageado, resolución dinámica de plugins (con ObjectProvider mejor), herramientas.

Preferir: inyección explícita siempre.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S206

RESPUESTA

NoUniqueBeanDefinitionException al arrancar (o al primer uso si lazy).

Resuelve: @Primary en uno, @Qualifier en el punto de inyección, o nombre de parámetro = nombre del bean (por nombre).

En @ConfigurationProperties-style: clases wrapper distintas para cada configuración.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S209

RESPUESTA

spring.main.lazy-initialization=true: TODOS los beans se crean perezosamente — arranque rapidísimo, pero errores de config aparecen tarde y primer request más lento.

@Lazy: quirúrgico por bean.

En prod se prefiere eager (fail-fast); lazy global útil para arranques en dev o lambdas con cold start.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S208

RESPUESTA

Método SOBRESERIBE el de la clase (regla de resolución: el más específico).

Métodos SIN anotación propia heredan la de la clase.

Beans internos (nueva clase anidada como @Component) Sí son beans con su propio proxy → transacción funciona.

Cuidado con clases de configuración anidadas no anotadas: son configuración, no beans.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S210

¿Cómo inyectarías un valor que cambia en runtime (refresh)?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S211

¿Cuál es la diferencia entre filtros de servlet y @WebFilter vs FilterRegistrationBean?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S212

¿Cómo arrancar código justo después de levantar el contexto?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S213

¿Cómo harías un feature flag en Spring sin librería externa?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S214

¿Qué es un 'fat jar' de Boot y por qué importa en contenedores?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S215

¿Diferencia entre @RequestParam, @RequestPart y @ModelAttribute para forms?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S216

¿Cómo proteger contra N+1 sin notarlo en dev?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S217

¿Cómo probarías que un @Scheduled corre con la frecuencia correcta?

INTERMEDIO

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

SPRING BOOT

Trampas de entrevista

S211

RESPUESTA

@WebFilter: anotación servlet nativa — requiere @ServletComponentScan y da poco control.
FilterRegistrationBean (recomendado en Boot): bean que registra el Filter con URL patterns, ORDEN explícito y activación condicional.
Orden importa: auth debe correr antes que logging de negocio; security filter chain corre antes de los tuyos si así lo ordenas.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S210

RESPUESTA

@ConfigurationProperties + @RefreshScope + actuador/refresh (Cloud Config) — o re-lectura del Environment.
Con Kubernetes: spring-cloud-kubernetes reloda ConfigMaps (polling/eventos).
Anti-patrón: @Value cacheado sin refresh (nunca cambia).
Para config MUY dinámica: feature flags service (LaunchDarkly/Unleash) o DB-backed properties con caché corta.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S213

RESPUESTA

@ConfigurationProperties(prefix="features") record Features(boolean nuevoCheckout, ...) + consumidor inyecta Features.
Toggle por properties/env → redeploy o refresh.
Con @ConditionalOnProperty para beans enteros (activar/desactivar implementaciones).
Nivel siguiente: flags runtime (Unleash/LaunchDarkly) sin redeploy, con % de usuarios.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S212

RESPUESTA

CommandLineRunner (args crudos) / ApplicationRunner (ApplicationArguments parseados) — beans ejecutados al final del arranque.
@EventListener(ApplicationReadyEvent.class): equivalente desacoplado.
SmartLifecycle para procesos con start/stop (consumers).
@PostConstruct NO: corre durante la creación del bean (contexto aún incompleto).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S215

RESPUESTA

@RequestParam: campo individual del form/query.
@ModelAttribute: objeto completo binded por setters (form-encode).
@RequestPart: parte de un MULTIPART con converter propio (JSON + archivo juntos).
Multipart simple: @RequestParam MultipartFile alcanza.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S214

RESPUESTA

Jar único con TODAS las dependencias (BOOT-INF/lib) + loader propio (java -jar directo).
Contra: rebuild = reenviar 80MB aunque solo cambió tu código → layered extraction resuelve (capas por cambio de dependencias).
Alternativa: war en Tomcat externo (legado).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S217

RESPUESTA

No esperes tiempos reales: extraer la lógica a un método y testarlo directo.
Para el wiring: Awaitility con ventana corta, o SchedulerTask de test con fixedDelay mínimo y count.
En cluster: verificar ShedLock (una sola ejecución con 2 instancias en test de integración).
Métricas de ejecución en prod (contador + timer).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S216

RESPUESTA

Tests de conteo: datasource-proxy o Hibernate statistics (assert selectCount == esperado).
hibernate.generate_statistics + log WARN de hibernate (session-metrics).
En prod: métricas por query, trace con spans por SELECT (Micrometer + JDBC observer).
Code review: toda relación navegada en loop = sospechoso.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S218

¿Cómo migrarías una propiedad de @Value a @ConfigurationProperties sin romper nada?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S219

¿Cómo expondrías 2 versiones de la misma API sin duplicar todo?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S220

¿Qué pasa si una ExceptionHandler lanza otra excepción?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S221

¿Cómo validar límites de rate por usuario SIN librerías?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S222

Un archivo de 500MB rompe el upload: config correcta

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S223

¿Diferencia entre @Scheduled y un Queue worker (Cloud Tasks)?

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S224

¿Cómo loguear el SQL real con parámetros en prod sin reventar performance?

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S225

flatMap vs concatMap y threading en WebFlux

AVANZADO

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

SPRING BOOT

Trampas de entrevista

S219

RESPUESTA

Núcleo único (service) + controllers por versión que mapean DTOs distintos: /v1 y /v2 comparten la lógica; transformadores versionados.
Solo duplicar lo que ROMPIÓ compatibilidad.
Deprecación: header Sunset + métricas de uso por versión para saber cuándo retirar v1.
Nunca: fork del service por versión.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S218

RESPUESTA

- 1) Crear el POJO con el campo igual (mismo nombre relaxed binding).
- 2) Registrar (@Component o @EnableConfigurationProperties).
- 3) Migrar consumidores uno a uno (compilación guía).
- 4) Tests de binding con ApplicationContextHolder.
- 5) Deprecar la clave vieja con placeholder de compatibilidad temporal: @Value("\${vieja:\${nueva:default}}").

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S221

RESPUESTA

Filtro/interceptor + Redis INCR con EXPIRE (ventana fija) o script Lua para sliding window: clave por userId+minuto.
429 con Retry-After; headers X-RateLimit-Limit/Remaining informativos.
Distribuido porque Redis comparte entre réplicas.
Producción seria: Bucket4j + Redis o el rate limiter del gateway.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S220

RESPUESTA

Spring cae al handler de último recurso (ErrorAttributes/BasicExceptionHandler) → 500 genérico; la excepción original se loguea por el stack.
El catch-all Exception.class del advice DEBE ser infalible: construir respuesta simple sin lógica que pueda fallar (y loguear AMBAS excepciones).
Nunca confiar en datos del request ahí (pueden ser la causa).

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S223

RESPUESTA

@Scheduled: periódico fijo, en-proceso, no distribuido sin lock, jobs 'batch'.
Queue worker (Tasks/PubSub): por-evento, con retry persistido, dead-letter, visibilidad del fallo, escala horizontal natural.
Regla: cron interno para mantenimiento simple; colas para trabajo que NO puede perderse.

INTERMEDIO

SPRING BOOT

Trampas de entrevista

S222

RESPUESTA

spring.servlet.multipart.max-file-size=600MB y max-request-size mayor; gateway/ingress también tiene su límite (client_max_body_size en nginx, timeout de LB).
Streaming: guardar el InputStream directo a disco/S3 (nunca byte[] completo).
Alternativa para archivos grandes: upload directo a Cloud Storage con signed URL — tu API solo firma y registra.

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S225

RESPUESTA

flatMap: ejecuta en PARALELO (intercala elementos según llegan) — para llamadas independientes.
concatMap: SECUENCIAL, respeta orden.
publishOn/subscribeOn cambian de scheduler (boundedElastic para código bloqueante: blockHound detecta violaciones).
Nunca bloquear el event loop: JDBC, sleep, CompletableFuture.get → boundedElastic.

AVANZADO

SPRING BOOT

Trampas de entrevista

S224

RESPUESTA

Nunca show-sql en prod. Opciones: datasource-proxy/psql wrapper con log SLOW queries (>200ms) + parámetros en DEBUG activado dinámicamente (actuador loggers level cambian SIN restart).
PG: pg_stat_statements / slow query log del motor.
Tracing: span por query con el statement (sampling) — equilibrando PII.

INTERMEDIO

SPRING BOOT

WebFlux y reactivo

S226

Backpressure en una frase + estrategias

AVANZADO

SPRING BOOT

WebFlux y reactivo

S227

R2DBC: JPA pero reactivo — qué cambia

AVANZADO

SPRING BOOT

WebFlux y reactivo

S228

SSE vs WebSocket vs Long-polling

AVANZADO

SPRING BOOT

★
Trampas de entrevista

S229

¿Qué mide y qué no
spring.jpa.open-in-view?

AVANZADO

SPRING BOOT

Trampas de entrevista

S230

Si debes garantizar que un email se envía
EXACTAMENTE una vez tras crear un
pedido, ¿qué haces?

AVANZADO

SPRING BOOT

Trampas de entrevista

S231

¿Cuándo un @Bean de @Configuration se
llama 'lite mode' y qué pierde?

AVANZADO

GCP

★
Fundamentos y organización

G001

Jerarquía de recursos en GCP

BASICO

GCP

★
Fundamentos y organización

G002

Presupuestos y optimización de costos en
GCP

BASICO

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

SPRING BOOT

WebFlux y reactivo

S227

RESPUESTA

spring-boot-starter-data-r2dbc + DatabaseClient/repositorios reactivos (ReactiveCrudRepository).

SQL no bloqueante real (driver por BD), transacciones con ReactiveTransactionManager.

Sin lazy loading (JOIN FETCH no existe): proyecciones/queries explícitas.

Menos conveniente que JPA; usar solo en pipelines reactivos.

AVANZADO

SPRING BOOT

WebFlux y reactivo

S226

RESPUESTA

El consumidor avisa cuántos elementos puede procesar (request(n)) para que el productor no lo ahogue.

Estrategias: buffer (acumular con límite), drop/latest (descartar), onError (abortar), limitRate en pipelines.

La cola con límite es backpressure aplicado a mensajes.

AVANZADO

SPRING BOOT



Trampas de entrevista

S229

RESPUESTA

OSIV (default TRUE): mantiene la sesión de Hibernate abierta durante TODO el request → los controllers pueden tocar relaciones lazy 'gratis'.

Problemas: conexiones del pool retenidas durante serialización (menos throughput), queries sorpresa en la vista, N+1 invisible.

Prod recomendado: false + DTOs/JOIN FETCH explícitos. Boot lo avisa al arrancar.

AVANZADO

SPRING BOOT

WebFlux y reactivo

S228

RESPUESTA

SSE (text/event-stream): unidireccional servidor→cliente, reconexión automática, HTTP puro — notificaciones, dashboards.

WebSocket: bidireccional full-duplex (STOMP sobre WS en Spring) — chat, edición colaborativa.

Long-polling: legado/fallback.

Escalar WS/SSE: sesiones pegajosas o pub/sub compartido (Redis) entre réplicas.

AVANZADO

SPRING BOOT

Trampas de entrevista

S231

RESPUESTA

@Bean en una clase SIN @Configuration (en @Component por ejemplo) = lite mode: los métodos NO pasan por proxy CGLIB → llamar un @Bean desde otro NO reutiliza el singleton (crea instancia nueva cada vez).

Con @Configuration completo: intercepta llamadas internas a métodos @Bean y devuelve el singleton.

Pierde inter-bean references — bug sutil y clásico.

AVANZADO

SPRING BOOT

Trampas de entrevista

S230

RESPUESTA

Exactamente una vez es un mito distribuido → lo asegurable: AL MENOS una vez + idempotencia.

Diseño: outbox (evento PedidoCreado en tabla, misma tx) → worker publica → consumidor envía email guardando dedup por eventId (si ya existe, skip).

Reintentos con backoff + dead letter para fallos persistentes.

AVANZADO

GCP



Fundamentos y organización

G002

RESPUESTA

Budgets con alertas (email/Pub/Sub) · labels para atribuir costo por equipo/app · informes de Billing.

Ahorro: committed use discounts (1-3 años), VMs Spot para carga interrumpible, apagar dev de noche, right-sizing recommendations, Cloud Run escala a 0.

BASICO

GCP



Fundamentos y organización

G001

RESPUESTA

Organization → Folders → Projects → Resources.

Los proyectos aíslan facturación, cuotas, IAM y APIs habilitadas.

Las políticas (IAM y org policies) se HEREDAN hacia abajo.

Buena práctica: un proyecto por entorno (dev/stage/prod) o dominio, carpetas por equipo.

BASICO

GCP

Fundamentos y organización

G003

¿Qué es un 'proyecto' en GCP?

BASICO

GCP

Fundamentos y organización

G004

Región vs zona vs multi-región: decisiones de arquitectura

BASICO

GCP

Fundamentos y organización

G005

Cloud Shell y Cloud Editor: qué te dan sin instalar nada

BASICO

GCP

Fundamentos y organización

G006

gcloud: configuración, proyecciones y comandos esenciales

BASICO

GCP

Fundamentos y organización

G007

Billing export: cómo saber quién gasta qué

BASICO

GCP

Fundamentos y organización

G008

Committed Use Discounts vs Spot VMs vs Sustained Use

BASICO

GCP★

Fundamentos y organización

G009

Recommender y right-sizing: cómo bajar la factura sin romper nada

BASICO

GCP

Fundamentos y organización

G010

Infraestructura como código: por qué Terraform y no console

BASICO

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

GCP

RESPUESTA

Fundamentos y organización

G004

Zona: datacenter físico (latencia mínima, riesgo de caída).
 Región: conjunto de zonas independientes (latencia baja entre zonas <2ms).
 Multi-región: conjunto de regiones — máxima disponibilidad y cercanía global (Cloud Storage multi-region, Spanner).
 Diseño: producción crítica = mínimo 2 zonas; global = multi-región para datos, LB global para tráfico.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G003

Unidad base de organización: dentro viven todos los recursos; define cuotas, facturación, IAM base y qué APIs están habilitadas. Cada recurso pertenece exactamente a un proyecto.
 Patrón habitual: proyectos separados por entorno (dev/stage/prod) bajo la misma organización.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G006

gcloud init / config set project ID; config list.
 Proyectos: gcloud projects create/list.
 Compute: gcloud compute instances list/create; Run: gcloud run deploy svc --image ... --region ...
 Formato: --format="value(name,zone)" o --format=json; --filter="status=RUNNING".
 Componentes: gcloud components update.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G005

Terminal Linux en el navegador con gcloud/gsutil/kubectl preinstalados y 5GB de home persistente.
 Editor web integrado; permite deploy de ejemplo en minutos. Útil: emergencias desde cualquier equipo, prototipos de APIs, administrar cuando no tienes el SDK local.
 Limitación: no para cargas largas (timeout de sesión).

BASICO

GCP

RESPUESTA

Fundamentos y organización

G008

Committed (CUD): 1-3 años por uso estable → hasta 57% off (paga compromiso de vCPU/RAM).
 Spot: hasta 91% off pero PUEDE ser evicted — solo cargas tolerantes a fallos (batch, workers con retry).
 Sustained use: descuento automático por uso mensual alto (solo en algunas regiones/máquinas).
 Estrategia: base en CUD, picos en estándar, batch en Spot.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G007

BigQuery export detallado: Billing > Export → tabla de línea por SKUs con labels, proyecto, servicios.
 Queries: costo por label team=api en el mes, tendencia día a día. Budgets con alertas (50/90/100%) y notificación a Pub/Sub para acciones automáticas.
 Recommender sugiere rightsizing y committed use.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G010

Console manual = estado invisible, irreproducible, sin code review, drift.
 Terraform: HCL declarativo, plan antes de apply, state como fuente de verdad, módulos reutilizables, CI/CD de infra.
 Google provider: google_compute_instance, google_cloud_run_service...
 Regla de oro: si no está en Terraform, no existe.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G009

Active Assist Recommender: sugiere instancias sobredimensionadas (p95 de CPU/mem), imágenes huérfanas, IPs sin usar.
 IDLE resources: discos sin attach, IPs estáticas sin usar (¡cobran!).
 Aplicar con métricas a mano: cambiar machine type requiere stop/start (o live migration en algunos casos).
 Reporte mensual de ahorros = credibilidad con finance.

BASICO

GCP

Fundamentos y organización

G011

Cloud Resource Manager y jerarquía por API

BASICO

GCP

Fundamentos y organización

G012

Soporte GCP y roles de emergencia (break-glass)

BASICO

GCP

★

Fundamentos y organización

G013

¿Qué es la nube? IaaS vs PaaS vs SaaS (con ejemplos GCP)

BASICO

GCP

Fundamentos y organización

G014

CapEx vs OpEx: por qué las empresas van a la nube

BASICO

GCP

Fundamentos y organización

G015

Firebase vs Google Cloud: ¿qué es y cuándo usar cada uno?

BASICO

GCP

Fundamentos y organización

G016

¿Cuántos proyectos y cómo organizarlos? (patrón de referencia)

INTERMEDIO

GCP

Fundamentos y organización

G017

Cuotas y límites: qué son y cómo gestionarlos

INTERMEDIO

GCP

Fundamentos y organización

G018

Labels vs tags vs nombres: qué usar para qué

INTERMEDIO

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

GCP

RESPUESTA

Fundamentos y organización

G012

Planos de soporte: Basic, Standard, Enhanced, Premium (SLA de respuesta).

Break-glass: cuenta admin de emergencia con credencial offline sellada, alertada al usarse, auditable (log de admin activity).

Runbooks con escalamiento y contactos de soporte versionados junto al código.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G011

APIs: projects.create, folders.create, setIamPolicy — automatiza el bootstrap de entornos.

Patrón: repo de infraestructura crea proyecto + APIs + budget + SA con Terraform; equipos solo despliegan apps.

Límites: 100 etiquetas? no — recursos por proyecto varían por servicio; API rate limits a considerar en tooling masivo.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G014

CapEx: comprar servidores (inversión grande adelantada, sobredimensionas para picos, te quedas corto igual).

OpEx: pagar por uso (cloud) — escala con la demanda, apagas lo que no usas, cero mantenimiento de hardware.

Otros motores: time-to-market (minutos vs semanas), regiones globales el mismo día, servicios gestionados (BD, ML) que un equipo chico no podría operar.

Contra: costo variable hay que GOBERNarlo (budgets, labels).

BASICO

GCP

★
RESPUESTA

Fundamentos y organización

G013

Alquilar cómputo gestionado por otro en lugar de comprar servidores.

IaaS (infraestructura): TÚ gestionas SO + apps — Compute Engine.

PaaS (plataforma): gestionas SOLO tu código — App Engine, Cloud Run.

SaaS (software): usas el producto terminado — Gmail, Workspace.

Regla mental: cuanto más 'aaS', menos infraestructura mantienes y más rápido entregas.

BASICO

GCP

RESPUESTA

Fundamentos y organización

G016

Por límite de cuota/aislamiento y por ciclo de vida: host proyecto de red (Shared VPC), un proyecto por entorno (dev/stage/prod) por servicio, proyectos de sistema (logging, billing, artifact registry).

Carpetas por equipo o por producto bajo la Organization.

Ventaja: blast radius pequeño, IAM granular, presupuestos por proyecto.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G015

Firebase: plataforma para apps MÓVILES y web — Auth, Firestore, Realtime DB, FCM (push), Hosting, Analytics. Pensada para desarrolladores de app, sin tocar infraestructura.

Google Cloud: infraestructura y servicios backend para CUALQUIER carga (VMs, Kubernetes, BigQuery...).

Se combinan: frontend móvil con Firebase Auth + backend API en Cloud Run + datos en Cloud SQL.

Firebase comparte infraestructura GCP (mismo IAM/Firestore).

BASICO

GCP

RESPUESTA

Fundamentos y organización

G018

Labels: pares key-value en recursos FACTURABLES — cost attribution (team=api, env=prod); heredables a Billing export.

Network tags: para FIREWALL rules de Compute (target-tags).

Resource name: identificador inmutable del recurso.

Gobernanza: org policy que exija labels obligatorios en instancias nuevas.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G017

Techo por recurso/servicio por proyecto-región (CPUs, IPs, API QPS) — protege de errores de costo y abuso.

Ver: gcloud compute project-info describe, console IAM > Quotas.

Pedir aumento: console quota request (algunos instantáneos).

Diseño: no dependas de cuotas altas sin alertas (403 QUOTA_EXCEEDED en producción duele).

INTERMEDIO

GCP

Fundamentos y organización

G019

Organization policies vs IAM policies

INTERMEDIO

GCP

Fundamentos y organización

G020

SLAs reales de los servicios: qué esperara tu arquitectura

INTERMEDIO

GCP

Fundamentos y organización

G021

Regiones: cómo elegir entre latencia, costo y residencia de datos

INTERMEDIO

GCP

Fundamentos y organización

G022

¿Qué es el Well Architecture Framework de GCP? (5 pilares)

INTERMEDIO

GCP

Fundamentos y organización

G023

Cloud APIs: cuotas, rate limits y manejo de errores

INTERMEDIO

GCP

Fundamentos y organización

G024

Cloud Marketplace y deployments terceros: riesgos

INTERMEDIO

GCP

Fundamentos y organización

G025

Resource hierarchy anti-patrones

INTERMEDIO

GCP

Fundamentos y organización

G026

Terraform state: por qué importa y buenas prácticas

AVANZADO

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

GCP

RESPUESTA

Fundamentos y organización

G020

Compute Engine 99.9% (instancia), multi-zona MIG 99.99%; Cloud Storage multi-region 99.95% (SLA de clase); Cloud Run 99.95% regional; Spanner 99.999% multi-región.

Tu SLA compuesto = producto de los SLAs de los componentes del path crítico.

Diseño por demanda: cada 9 extra cuesta complejidad — no todo necesita 99.99%.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G019

IAM: QUIÉN puede hacer QUÉ (permisos, aditivo).

Org policy: QUÉ se permite hacer EN GENERAL (restricciones):

solo IPs externas permitidas, solo ubicaciones específicas

(gcp.resourceLocations), deshabilitar service account key creation, VMs sin IP pública.

Se heredan por la jerarquía y aplican a TODOS (incluso owners) — guardrails de compliance.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G022

Operational excellence (IaC, observabilidad, automatización), Security (least privilege, encryption, defense in depth), Reliability (HA, DR, quotas), Performance (escalar, caching, regiones), Cost optimization (CUDs, right-size, apagar).

Usarlo como checklist de revisión de arquitectura antes de lanzar.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G021

Prioridad: 1) requisitos legales de residencia (PII en país X), 2) latencia a usuarios, 3) costo (regiones caras vs baratas hasta 2x), 4) catálogo de servicios disponible.

Herramientas: ping/latency de Cloud CDNs, pricing calculator, página de regiones por servicio.

Multi-región para datos + routing global resuelve mayoría de casos.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G024

Marketplace: soluciones empaquetadas (DBs partners, software comercial) desplegadas en tu proyecto con billing integrado.

Riesgos: revisar qué Service Accounts y permisos pide, costos de licencia mensual, soporte del vendor.

Para POCs acelera; para core business evaluar antes el managed nativo.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G023

Cada API: límites de QPS por proyecto. 429/403

RESOURCE_EXHAUSTED → exponential backoff + retry con jitter (client libs lo hacen por defecto).

Errores gRPC comunes: UNAVAILABLE (retry transitorio),

PERMISSION_DENIED (IAM), FAILED_PRECONDITION (APIs no habilitadas).

Cliente lib vs REST puro: libs manejan auth/retries/ batching.

INTERMEDIO

GCP

RESPUESTA

Fundamentos y organización

G026

State guarda el mapeo recursos-reales ↔ código.

Remoto + lockeado: bucket GCS con versioning (backend "gcs") — evita corruption y race conditions en equipo.

Nunca editar a mano; import para recursos existentes (terraformer/import block).

State contiene secretos → bucket privado + acceso controlado.

AVANZADO

GCP

RESPUESTA

Fundamentos y organización

G025

Un solo proyecto gigante para todo (cuotas, blast radius, permisos groseros).

Un proyecto por VM/servicio mínimo (operación imposible).

Carpetas por TIPO de recurso en vez de por equipo/producto (IAM no escala).

Entornos mezclados en el mismo proyecto (dev junto a prod = desastre).

Regla: cambiar algo de dev NO debe poder tocar jamás a prod.

INTERMEDIO

GCP

Fundamentos y organización

G027

Disaster recovery: RTO/RPO y niveles de estrategia GCP

AVANZADO

GCP

Fundamentos y organización

G028

Migración de cuentas y organización enterprise: Landing Zone

AVANZADO

GCP

★

IAM y seguridad

G029

¿Cómo funciona IAM en GCP?

BASICO

GCP

IAM y seguridad

G030

Service Accounts: ¿qué son y cómo se usan bien?

INTERMEDIO

GCP

★

IAM y seguridad

G031

¿Para qué sirve Secret Manager?

INTERMEDIO

GCP

IAM y seguridad

G032

Autenticación de usuarios finales vs de servicios

INTERMEDIO

GCP

★

IAM y seguridad

G033

¿Qué es Application Default Credentials (ADC)?

INTERMEDIO

GCP

IAM y seguridad

G034

Custom roles: cuándo valen la pena

INTERMEDIO

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

GCP

Fundamentos y organización

G028

RESPUESTA

Landing zone: la base lista antes de crear proyectos — Organization, carpetas, org policies base (no public IPs, no SA keys), jerarquía de billing, Shared VPC host, logging central, Terraform bootstrap (seed project + state bucket + CI). Blueprints: module terraform-google-project-factory. Sin landing zone, cada equipo inventa su seguridad — deuda permanente.

AVANZADO

GCP

Fundamentos y organización

G027

RESPUESTA

RTO: tiempo para recuperar; RPO: datos máximos a perder. Niveles: backup & restore (horas/días), pilot light (BD replicada, cómputo apagado — minutos), warm standby (réplica chica corriendo), hot/active-active (multi-región — segundos). Herramientas: snapshots cross-region de Cloud SQL, GCS dual/multi-region, Spanner multi-región, DNS failover. Probar el DR (gameday) o no existe.

AVANZADO

GCP

IAM y seguridad

G030

RESPUESTA

Identidad para CÓDIGO, no personas (email: sa@proyecto.iam.gserviceaccount.com). A cada workload (Cloud Run, GKE, Compute) le adjuntas una SA con roles mínimos; la app autentica con ADC sin claves. Evita descargar JSON keys; si son inevitables: rotar y guardar en Secret Manager. Impersonation (actAs) para tareas puntuales.

INTERMEDIO

GCP



IAM y seguridad

G029

RESPUESTA

Policy = QUIÉN (member) con qué ROL sobre qué RECURSO. Members: usuarios, grupos, service accounts. Roles: básicos (Owner/Editor/Viewer, muy amplios), predefinidos (granulares: roles/pubsub.publisher), custom. Se hereda por la jerarquía org → carpeta → proyecto. Principio: least privilege.

BASICO

GCP

IAM y seguridad

G032

RESPUESTA

IAM autentica administradores y service accounts. Para usuarios FINALES de tu app: Identity Platform / Firebase Auth (email, Google, OAuth federado) emite JWTs que tu backend valida (firma + audience). API Gateway / Cloud Endpoints pueden validar JWTs antes de llegar a tu servicio.

INTERMEDIO

GCP



IAM y seguridad

G031

RESPUESTA

Bóveda central de secretos (API keys, passwords, tokens) con versionado, IAM por secreto, rotación, auditoría y replicación. La app lee secretos en runtime con la client library usando ADC. Nada de secretos hardcoded ni en variables de entorno planas.

INTERMEDIO

GCP

IAM y seguridad

G034

RESPUESTA

Cuando un predefinido es muy amplio y necesitas solo un subconjunto: roles/custom.verOnlyRun = compute.instances.get + list... Límite 300 custom por org; mantenerlos documentados y revisados. Alternativa moderna: reusar predefinidos pequeños en vez de crear custom por equipo (gobernanza más simple).

INTERMEDIO

GCP



IAM y seguridad

G033

RESPUESTA

Mecanismo estándar de autenticación: las client libraries buscan credenciales en orden:
1. GOOGLE_APPLICATION_CREDENTIALS (archivo)
2. SA adjunta al entorno (metadata server en Cloud Run/GKE/Compute)
3. credenciales de gcloud en local.
Así el MISMO código corre en local y en GCP sin claves en el código.

INTERMEDIO

GCP

IAM y seguridad

G035

Audit logs: qué registra GCP por ti

INTERMEDIO

GCP

IAM y seguridad

G036

Claves de Service Account: por qué evitarlas y qué hacer si son inevitables

INTERMEDIO

GCP

IAM y seguridad

G037

Cloud Data Loss Prevention (Sensitive Data Protection): para qué sirve

INTERMEDIO

GCP

IAM y seguridad

G038

Defensa en profundidad en GCP: las 6 capas

INTERMEDIO

GCP

IAM y seguridad

G039

Rotación de secretos sin downtime: patrón completo

INTERMEDIO

GCP

IAM y seguridad

G040

Anatomía de una IAM policy binding

AVANZADO

GCP

IAM y seguridad

G041

Roles predefinidos más usados que debes reconocer

AVANZADO

GCP

IAM y seguridad

G042

IAM Conditions: permisos temporales o por recurso

AVANZADO

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

GCP

IAM y seguridad

G036

RESPUESTA

Riesgo: credencial larga que puede filtrarse (git, image, laptop), rotación manual, sin origen.

Alternativas: ADC/workload attached SA (GCP), Workload Identity Federation (externos), impersonation (CLI).

Si inevitable: org policy permite bloquear creación, Secret Manager para guardar, rotación <90 días, scope mínimo, detección de uso (SCC anomaly).

INTERMEDIO

GCP

IAM y seguridad

G035

RESPUESTA

Admin Activity (siempre, quién creó/modificó qué) + Data Access (lecturas de datos, opt-in por servicio) + System Event + Policy Denied.

Exportar a BigQuery para búsquedas tipo 'quién descargó el bucket X en marzo'.

Retención default 400 días (admin), configurable a más con bucket lock.

INTERMEDIO

GCP

IAM y seguridad

G038

RESPUESTA

- 1) Identidad: IAM mínimo, MFA, Federation.
- 2) Perímetro: VPC-SC, org policies, Armor.
- 3) Red: private IPs, firewall deny-by-default, NAT controlado.
- 4) Datos: cifrado (CMEK), DLP, retention.
- 5) Aplicación: secretos, image scanning, Binary Authorization.
- 6) Operación: SCC, audit logs, alertas.

Ninguna capa sola basta; el ataque se detiene en el anillo que no esperaba.

INTERMEDIO

GCP

IAM y seguridad

G037

RESPUESTA

Detectar y clasificar PII en texto/imágenes/tablas: emails, tarjetas, DNI, nombres.

De-identify: masking, tokenización, hashing (format-preserving).

Casos: escanear buckets/BigQuery en busca de datos sensibles expuestos, anonimizar logs antes de guardar, minimizar datos en analítica.

API por contenido o job scan programado.

INTERMEDIO

GCP

IAM y seguridad

G040

RESPUESTA

bindings: [{role: roles/run.invoker, members: [serviceAccount:sa@p.iam..., group:equipo@x.com, user:ana@x.com]}]

Aplica sobre un recurso (proyecto, bucket, servicio) y SE HEREDA hacia abajo.

Auditoría: getIamPolicy; modificar con etag (optimistic concurrency).

Deny policies (nuevo): reglas explícitas de denegación que SIEMPRE ganan sobre allow.

AVANZADO

GCP

IAM y seguridad

G039

RESPUESTA

- 1) Secret Manager guarda versión N; app cachea con TTL.
- 2) Rotación: crear versión N+1 (doble válido en BD si es password) → esperar TTL de cache → invalidar N.
- 3) Automatizar: Secret Manager rotation schedule + Cloud Function/Run que cambia la BD y publica evento.

Conexiones vivas viejas: drain con max lifetime de pool < período de rotación.

INTERMEDIO

GCP

IAM y seguridad

G042

RESPUESTA

Condición CEL en el binding:

resource.name.startsWith("projects/_/buckets/logs-prod"), request.time < timestamp("2025-12-31T00:00:00Z") (acceso temporal automático).

Casos: acceso solo a un bucket, contractor con fecha de expiración, entornos por sufijo de nombre.

Ojo: condiciones solo en algunos niveles (proyecto y recursos que soportan conditions).

AVANZADO

GCP

IAM y seguridad

G041

RESPUESTA

roles/viewer, roles/editor, roles/owner (básicos — evitar en prod).

Por servicio: roles/run.invoker (invocar), roles/run.admin, roles/storage.objectViewer/objectAdmin, roles/cloudsql.client, roles/secretmanager.secretAccessor, roles/pubsub.publisher/subscriber, roles/logging.logWriter.

Elije el mínimo que cumpla la tarea.

AVANZADO

GCP

IAM y seguridad

G043

Workload Identity Federation: autenticar CI/CD SIN claves

AVANZADO

GCP

IAM y seguridad

G044

Service Account Hierarchy: user-managed vs default SAs

AVANZADO

GCP

IAM y seguridad

G045

actAs y impersonation: delegar sin dar tus permisos

AVANZADO

GCP

IAM y seguridad

G046

Cloud KMS y CMEK: claves de cifrado propias

AVANZADO

GCP

IAM y seguridad

G047

VPC Service Controls: el perímetro anti-exfiltración

AVANZADO

GCP

IAM y seguridad

G048

Cloud Armor: WAF y anti-DDoS en el edge

AVANZADO

GCP

IAM y seguridad

G049

Identity-Aware Proxy (IAP): acceso administrativo sin VPN

AVANZADO

GCP

IAM y seguridad

G050

Security Command Center: el dashboard de seguridad

AVANZADO

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

GCP

IAM y seguridad

G044

RESPUESTA

Default SAs (compute@..., run-invoker): permisos amplios (editor) — riesgo.
Práctica: una SA POR SERVICIO con permisos mínimos (api-sa@proj.iam), attach al deploy, y deshabilitar/restringir default SAs vía org policy.
Nomenclatura: nombre corto + entorno (pedidos-prod@...).

AVANZADO

GCP

IAM y seguridad

G043

RESPUESTA

GitHub Actions/Azure/Okta asume una identidad externa → intercambio por token GCP temporal vía provider (pool + provider con mapping de claims).
Cero JSON keys descargadas, sin rotación manual, audit trail.
Config: gcloud iam workload-identity-pools create + add-azure/github provider + bind SA con roles/iam.workloadIdentityUser.

AVANZADO

GCP

IAM y seguridad

G046

RESPUESTA

Por default Google cifra at-rest con claves de Google (Google-managed).
CMEK: TU keyring/clave en Cloud KMS protege Cloud SQL, GCS, BigQuery, Pub/Sub, disks... — se puede deshabilitar la clave y los datos quedan inaccesibles (kill switch).
Rotación programada; HSM protection level para compliance alto.
Costo por clave + versión.

AVANZADO

GCP

IAM y seguridad

G045

RESPUESTA

roles/iam.serviceAccountTokenCreator sobre una SA permite GENERAR tokens como ella (gcloud auth impersonate-credentials, SDK Option.withCredentials).
Humanos con acceso a CLI operan 'como' la SA sin conocer sus claves.
Todo queda auditado como el token impersonado + el actor.

AVANZADO

GCP

IAM y seguridad

G048

RESPUESTA

Security policies adjuntas al backend service del LB: reglas preconfiguradas (XSS, SQLi, LFI), rate-based bans (N req/min por IP), geo-restricciones, allow/deny/redirect.
Adaptive Protection (ML) contra ataques volumétricos L3/L4 en Cloud Armor Enterprise.
Reglas en yaml/terraform versionadas.

AVANZADO

GCP

IAM y seguridad

G047

RESPUESTA

Perímetro lógico alrededor de servicios (GCS, BigQuery, Secret Manager...): solo tráfico desde proyectos/redes del perímetro.
Bloquea copiar data a un bucket externo aunque IAM lo permita.
Dry-run primero (audit) para no romper, luego enforce.
Concepto clave para compliance de datos sensibles.

AVANZADO

GCP

IAM y seguridad

G050

RESPUESTA

SCC: inventario y hallazgos (findings) de vulnerabilidades (VM vulnerable, bucket público, SA con clave vieja), threat detection (anomalías), posture management (compliance CIS).
Integración con Pub/Sub para automatizar remediación.
Standard vs Premium/Enterprise (más detectores y SIA).

AVANZADO

GCP

IAM y seguridad

G049

RESPUESTA

IAP delante de apps internas/admin: el usuario pasa por Google login y el LB inyecta el header X-Goog-Authenticated-User — solo members con roles/iap.httpsResourceAccessor pasan.
Para SSH/RDP a VMs: IAP Desktop Tunneling sin IP externa ni bastión.
Zero-trust interno: identidad por request.

AVANZADO

GCP

IAM y seguridad

G051

Binary Authorization: solo imágenes firmadas se despliegan

AVANZADO

GCP

IAM y seguridad

G052

Cloud Armor rate limiting y bot management

AVANZADO

GCP

★

Redes

G053

VPC, subredes y firewall rules

INTERMEDIO

GCP

★

Redes

G054

Cloud Load Balancing y Cloud CDN

INTERMEDIO

GCP

★

Redes

G055

¿Cómo conecto Cloud Run/GKE con recursos privados?

INTERMEDIO

GCP

★

Redes

G056

VPC global vs subredes regionales: entendiendo el modelo GCP

AVANZADO

GCP

Redes

G057

Firewall rules: campos y orden de evaluación

AVANZADO

GCP

★

Redes

G058

Cloud NAT: por qué casi todo backend lo necesita

AVANZADO

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

GCP

IAM y seguridad

G052

RESPUESTA

Reglas rate-based: throttle por IP (N req/min), ban temporal al exceder; por cookie/header con Expression language.
Bot management (Enterprise): reCAPTCHA Enterprise integrado — score por request, desafíos.
Geo + ASN blocking para mercados definidos.
Métricas de la policy → detectar ataques en curso antes del SLO burn.

AVANZADO

GCP

IAM y seguridad

G051

RESPUESTA

Policy que exige attestations (firma del pipeline de CI) antes de permitir deploy en GKE/Cloud Run.
Chainguard/Sigstore cosign para firmar; provenance de Cloud Build como attestor.
Bloquea supply chain attacks (imagen 'cualquiera' en producción).
Dry-run mode para calibrar sin romper.

AVANZADO

GCP



Redes

G054

RESPUESTA

Balanceador global anycast HTTP(S) L7 (o L4): terminación SSL, URL maps que enrutan por path/host a backend services (VMs, serverless NEG, Storage).
Cloud CDN cachea en edge POPs cerca del usuario.
Cloud Armor añade WAF y protección anti-DDoS.

INTERMEDIO

GCP



Redes

G053

RESPUESTA

VPC: red privada GLOBAL; subredes REGIONALES con rangos CIDR.
Firewall: reglas stateful allow/deny por prioridad, aplican a targets (tags de red o service accounts).
Conectar VPCs: VPC Peering o Shared VPC.
Private Google Access: llegar a las APIs de Google sin IP externa.

INTERMEDIO

GCP



Redes

G056

RESPUESTA

Una VPC es GLOBAL (abarca todas las regiones); las SUBREDES son regionales con CIDR propio.
Dentro de la VPC, tráfico inter-región viaja por el backbone de Google (cifrado en tránsito).
Comparado con AWS (VPC por región): menos piezas para conectar multi-región.
Auto-mode VPC crea subredes default — para prod usa custom mode.

AVANZADO

GCP



Redes

G055

RESPUESTA

Serverless VPC Access connector (Cloud Run/Functions) para alcanzar la VPC privada: VMs, Memorystore, Cloud SQL con IP privada.
GKE habla nativamente con la VPC.
Cloud SQL Auth Proxy o conectores IAM para bases de datos.
Private Service Connect: consumir servicios gestionados por IP privada.

INTERMEDIO

GCP



Redes

G058

RESPUESTA

VMs/GKE con SOLO IP privada necesitan salir a internet (actualizar paquetes, llamar APIs externas): Cloud NAT provee egress NAT gestionado (IPs estáticas de salida).
Beneficios: sin IPs públicas (superficie de ataque), IP fija para allowlists del tercero.
Se asocia por router/subred; ports por VM ajustables para muchas conexiones.

AVANZADO

GCP

Redes

G057

RESPUESTA

Reglas stateful: direction (ingress/egress), source/dest (CIDR, tags, service accounts), protocolo/puerto, action allow/deny, PRIORITY (0-65535, MENOR gana), disabled.
Default: deny all ingress, allow all egress.
Targeting por tags de red o por service account (más preciso).
Logs de firewall activables por regla.

AVANZADO

GCP

Redes

G059

Shared VPC vs VPC Peering: cuándo cada uno

AVANZADO

GCP

Redes

G060

Cloud Load Balancing: mapa de tipos

AVANZADO

GCP

★

Redes

G061

Health checks del LB: por qué tu 200 vale menos de lo crees

AVANZADO

GCP

Redes

G062

Cloud DNS y dominios: gestión básica

AVANZADO

GCP

Redes

G063

Cloud Interconnect vs Cloud VPN: conectividad híbrida

AVANZADO

GCP

Redes

G064

Network tiers y egress: cómo se cobra la salida de datos

AVANZADO

GCP

Redes

G065

Private Google Access y Private Service Connect: APIs sin internet

AVANZADO

GCP

Redes

G066

Serverless VPC Access connector: puente Cloud Run → VPC

AVANZADO

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

GCP

Redes

G060

RESPUESTA

Global externo: HTTPS LB (L7, anycast, Cloud Armor/CDN), SSL Proxy/TCP Proxy (L4 no-HTTP), External passthrough Network LB (UDP/TCP directo).

Regional: internal HTTP(S), internal passthrough (entre servicios de la VPC).

Serverless NEG: LB delante de Cloud Run/Functions con dominio propio.

Componentes: forwarding rule → target proxy → URL map → backend service → NEG/MIG.

AVANZADO

GCP

Redes

G059

RESPUESTA

Shared VPC: UNA red central administrada por el equipo de plataforma (host project) donde los proyectos de servicio despliegan sin tocar red — control centralizado estricto.

Peering: conecta VPCs independientes 1:1 (transitividad NO incluida) — equipos autónomos.

Enterprise: Shared VPC para prod; peering para integraciones puntuales.

AVANZADO

GCP

Redes

G062

RESPUESTA

Zones públicas (tu dominio) y privadas (nombres internos svc.internal → IPs privadas).

DNS peering/forwarding para resolver hacia on-prem u otras VPCs.

DNSSEC activable; TTL bajo (60s) antes de migraciones.

Routing policies: geolocation/failover — DNS como primer nivel de DR.

AVANZADO

GCP ★

Redes

G061

RESPUESTA

El health check del backend service determina si la instancia recibe tráfico — debe chequear un endpoint REAL (ping a DB ligero ok, pero pesado = cascada de reinicios).

Firewall: permitir 35.191.0.0/16 y 130.211.0.0/22 (rangos del LB).

Peligro: health check que depende de una dependencia lenta → app sana marcada DOWN → outage por diseño.

AVANZADO

GCP

Redes

G064

RESPUESTA

Premium tier: sale por backbone de Google (óptimo global).

Standard tier: sale por ISP regional — más barato, latencia variable.

Egress es el costo invisible que crece: streaming de archivos/videos a usuarios lejanos = dinero.

CDN + caché reduce egress drásticamente; medir en Billing por SKU.

AVANZADO

GCP

Redes

G063

RESPUESTA

VPN (HA VPN): túneles IPsec sobre internet — días de setup, costo bajo, 99.99% con 2 interfaces; para inicio de híbrido.

Dedicated/Partner Interconnect: circuito físico privado (10/100Gbps o partner) — latencia estable, más caro, para workloads serios.

Router Cloud (BGP) intercambia rutas en ambos.

Patrón: empezar VPN, migrar a Interconnect al crecer.

AVANZADO

GCP

Redes

G066

RESPUESTA

El connector (red por región, rango /28) permite que Cloud Run/Functions alcancen recursos de VPC privada: VMs, Memorystore, Cloud SQL con IP interna.

Costo por instancia del connector + throughput; dimensionar (2-10 instancias min/max según tráfico).

Direct VPC egress (nuevo, mejor): sin connector, salida directa por subred.

AVANZADO

GCP

Redes

G065

RESPUESTA

PGA: subred con ruta a 'private.googleapis.com' → las apps llaman APIs de Google (GCS, Secret Manager) por IP PRIVADA sin NAT ni internet.

PSC: endpoints privados para servicios específicos (incluye terceros en Marketplace y tu propio servicio detrás de un LB interno).

Estricto zero-internet: ambos + Cloud NAT solo si es indispensable.

AVANZADO

GCP

Redes

G067

Packet Mirroring y Flow Logs: depuración de red

AVANZADO

GCP

Redes

G068

Troubleshooting de conectividad GCP: el árbol de decisión

AVANZADO

GCP

Redes

G069

TLS interno entre servicios: mTLS con service mesh en GCP

AVANZADO

GCP

★

Compute y contenedores

G070

Compute Engine vs App Engine vs Cloud Run vs GKE

BASICO

GCP

★

Compute y contenedores

G071

¿Cuándo elegirías Cloud Run?

BASICO

GCP

Compute y contenedores

G072

Machine types: familias y cuál elegir

BASICO

GCP

Compute y contenedores

G073

Disks en Compute Engine: tipos y cuándo

INTERMEDIO

GCP

Compute y contenedores

G074

Managed Instance Groups: autoscaling y autohealing

INTERMEDIO

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

GCP

Redes

G068

RESPUESTA

1) ¿La VM/pod tiene IP esperada y rutas? (gcloud compute routes list)
 2) Firewall: regla allow aplicando al target correcto (tags/SA) y source correcto.
 3) Para servicios gestionados: Private Google Access / PSC / connector configurado?
 4) DNS: zona privada resolviendo el FQDN interno?
 5) Flow logs + firewall rules logging = evidencia del paquete denegado.
 Connectivity Tests (Network Intelligence) simula el path y dice dónde se corta.

AVANZADO

GCP

Redes

G067

RESPUESTA

VPC Flow Logs: metadatos de conexiones (5-tuple, bytes, aceptado/denegado) exportables a BigQuery — responder '¿quién habla con quién y cuánto?'.
 Firewall rules logging: qué regla bloqueó qué.
 Packet Mirroring: copia completa de paquetes a un collector (IDS/IPS como Suricata) para análisis profundo.
 Ante 'no conecta': flow logs primero, mirroring después.

AVANZADO

GCP ★

Compute y contenedores

G070

RESPUESTA

Compute Engine: VMs (IaaS), control total — lift & shift.
 App Engine: PaaS clásico (standard/flexible).
 Cloud Run: contenedor serverless, escala a 0, pagas por uso.
 GKE: Kubernetes gestionado para orquestar muchos microservicios.
 Serverless simple → Cloud Run; control de clúster → GKE.

BÁSICO

GCP

Redes

G069

RESPUESTA

Istio (Anthos Service Mesh) en GKE: sidecars/ambient manejan mTLS automático entre workloads con identidades SPIFFE — sin tocar código.
 Autorización L7: AuthorizationPolicy (frontend puede llamar pedidos: GET, nada más).
 Cloud Run/GKE sin mesh: TLS termina en LB; interno confía en red privada (menos fuerte).
 Zero trust interno: mesh es la respuesta estándar.

AVANZADO

GCP

Compute y contenedores

G072

RESPUESTA

E2: costo mínimo (no garantiza hardware fijo) — dev, web ligera.
 N2/N2D: propósito general balanceado (la default de prod).
 C2/C3: compute optimizado (CPU puro). M2/M3: memoria enorme. A2/A3: GPU (ML).
 Custom shapes: vCPU/RAM a la carta en N2.
 Regla: empezar N2D estándar, medir, ajustar; Recommender sugiere.

BÁSICO

GCP ★

Compute y contenedores

G071

RESPUESTA

Cuando empaquetas la app en un contenedor Docker sin gestionar infraestructura: APIs HTTP, webhooks, microservicios, eventos.
 Escala automático a 0 (no pagas en reposo) hasta N instancias; concurrencia por instancia configurable.
 Debe ser stateless y escuchar en la variable \$PORT.

BÁSICO

GCP

Compute y contenedores

G074

RESPUESTA

MIG: grupo de VMs idénticas desde instance template.
 Autoscaling por CPU/utilización o métrica de LB; autohealing con health check (recrea VMs malas); rolling updates (surge/unavailable) y stateful MIGs.
 Zonal (una zona) vs Regional (3 zonas — HA por diseño).
 Es 'Kubernetes sin Kubernetes' para apps simples.

INTERMEDIO

GCP

Compute y contenedores

G073

RESPUESTA

Standard PD: barato, secuencial (backups). Balanced PD: default razonable. SSD PD: IOPS/throughput alto (DBs). Hyperdisk: máximo performance desacoplada del tamaño.
 Snapshots: incrementales multi-region — backups programados (snapshot schedule).
 Regional PD: replicación sincrónica entre zonas para BDs auto-alojadas HA.

INTERMEDIO

GCP

Compute y contenedores

G075

Instance template + startup script: boot reproducible

INTERMEDIO

GCP

Compute y contenedores

G076

Spot VMs: patrón de uso correcto

INTERMEDIO

GCP

Compute y contenedores

G077

OS Login y metadata server: acceso y datos de instancia

INTERMEDIO

GCP

★

Compute y contenedores

G078

GKE Autopilot vs Standard: decisión práctica

INTERMEDIO

GCP

Compute y contenedores

G079

Node pools y taints/tolerations: aislar cargas

INTERMEDIO

GCP

★

Compute y contenedores

G080

Workload Identity en GKE: pods con SA de Google sin claves

INTERMEDIO

GCP

Compute y contenedores

G081

HPA y VPA: autoscaling de pods y nodos

INTERMEDIO

GCP

Compute y contenedores

G082

Container Registry → Artifact Registry: el switch obligatorio

INTERMEDIO

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

GCP

Compute y contenedores

G076

RESPUESTA

Hasta 91% de descuento; Google puede EVICTAR (preemption notice de 30s).
 Ideal: workers de colas (Pub/Sub redeliver), batch, CI runners, render — todo con checkpoint/retry.
 NO para: API crítica, BDs, estado sin replicar.
 Combina con MIG (mezcla estándar + spot) para absorber picos baratos.

INTERMEDIO

GCP

Compute y contenedores

G075

RESPUESTA

Template: imagen, machine type, network, SA, metadata — immutable; cambios = template nuevo + MIG rolling.
 Startup script (metadata startup-script): bash al boot — instalar paquetes, tocar configs.
 Mejor: imagen dorada pre-baked (Packer/Cloud Build) + startup script mínimo = arranque rápido y determinista.

INTERMEDIO

GCP ★

Compute y contenedores

G078

RESPUESTA

Autopilot: Google gestiona nodos (pagas por pods) — menos ops, seguridad por default (no root, hardened), SLA de pods; costo ligeramente mayor y menos control (taints, kernel).
 Standard: control total de node pools, DaemonSets, costos más bajos a gran escala — requiere equipo k8s.
 Default moderno: Autopilot salvo necesidad específica.

INTERMEDIO

GCP

Compute y contenedores

G077

RESPUESTA

OS Login: SSH por permisos IAM (roles/compute.osLogin) — sin gestionar keys por VM, 2FA heredado, audit por usuario.
 Metadata server (169.254.169.254): la instancia lee su SA token, zona, custom metadata — así funcionan ADC sin claves.
 Bloquear metadata con headers? No aplica (Google usa `Metadata-Flavor`).

INTERMEDIO

GCP ★

Compute y contenedores

G080

RESPUESTA

1) KA SA de Google + binding roles/iam.workloadIdentityUser al K8s SA.
 2) K8s SA anotada con iam.gke.io/gcp-service-account.
 3) Pod usa el K8s SA → metadata server entrega tokens de la Google SA.
 Reemplaza key.json montados — el estándar de acceso a GCP desde GKE.

INTERMEDIO

GCP

Compute y contenedores

G079

RESPUESTA

Node pool por tipo de carga: pool GPU (taint nvidia.com/gpu), pool spot, pool con máquina de memoria.
 Taint en el pool + toleration en el pod = solo ciertos pods aterrizan ahí.
 nodeSelector/affinity para ubicación (zona, tipo).
 Auto-upgrade por release channel (regular/stable) con ventanas de mantenimiento.

INTERMEDIO

GCP

Compute y contenedores

G082

RESPUESTA

Container Registry (gcr.io) deprecado; Artifact Registry (REGION-docker.pkg.dev/PROJECT/REPO/IMAGE).
 Repos tipados: docker, npm, maven, python, apt... por región (más cerca = pull más rápido y sin egress).
 Remote repos (proxy a Docker Hub) y vulnerabilidad scanning integrado.
 gcloud auth configure-docker region-docker.pkg.dev.

INTERMEDIO

GCP

Compute y contenedores

G081

RESPUESTA

HPA: replica PODS por CPU/mem o métricas custom (QPS de Prometeo) — la primera línea.
 VPA: recomienda/ajusta requests-limits por pod (right-sizing; modo recomendación en prod).
 Cluster Autoscaler: añade/elimina NODOS cuando no caben pods o sobran.
 KEDA para scale por longitud de cola (event-driven).

INTERMEDIO

GCP

Compute y contenedores

G083

GKE networking: VPC-native (alias IPs) y por qué importa

INTERMEDIO

GCP

Compute y contenedores

G084

Release channels y upgrades de GKE sin sustos

INTERMEDIO

GCP

Compute y contenedores

G085

Cloud SQL como servicio en GKE: proxy sidecar vs connector

INTERMEDIO

GCP

Compute y contenedores

G086

Ingress vs Gateway API vs Service de tipo LoadBalancer en GKE

AVANZADO

GCP

Compute y contenedores

G087

Network Policy en GKE: microsegmentación

AVANZADO

GCP

Compute y contenedores

G088

Dataplane V2 y Network Policies: rendimiento + seguridad

AVANZADO

GCP

Compute y contenedores

G089

StatefulSet, PVs y discos en GKE

AVANZADO

GCP

Compute y contenedores

G090

Node auto-provisioning y bin packing: optimizar costo de clúster

AVANZADO

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

GCP

Compute y contenedores

G084

RESPUESTA

Channels: rapid/regular/stable — Google parchea nodos y control plane automáticamente.

Estrategia: clúster de stage en rapid para ver el futuro; prod en regular.

Maintenance windows y exclusions para no parchear en picos.

PodDisruptionBudgets + surge upgrades: la app sobrevive al drenaje de nodos.

INTERMEDIO

GCP

Compute y contenedores

G083

RESPUESTA

VPC-native: pods reciben IPs DE LA SUBRED (secondary range) → visibles/ruteables en la VPC, firewall por pod, mejor para service mesh y politiques.

Routes-based (legado): NAT de IP de nodo — rompe firewalls por pod.

Requisito para Shared VPC, Cloud NAT por pod, Network Policies con Dataplane V2.

Pools nuevos ya son VPC-native por default.

INTERMEDIO

GCP

Compute y contenedores

G086

RESPUESTA

Service LoadBalancer: IP externa por servicio (caro, uno por svc).

Ingress (GCE o nginx controller): rutas HTTP/HTTPS por host/path a Services — el clásico.

Gateway API (el sucesor): roles separados (Gateway vs HTTPRoute), portable entre vendors, traffic split nativo.

Nuevo diseño: Gateway API; Ingress sigue soportado.

AVANZADO

GCP

Compute y contenedores

G085

RESPUESTA

Sidecar del Auth Proxy (contenedor junto a tu app): conexiones locales por socket unix, IAM auth, cifrado — el patrón estándar.

En Autopilot: workload identity + proxy con recursos asignados.

Direct private IP + gcp cloud-sql-proxy como DaemonSet para muchas apps.

Pool sizing: cada instancia de app abre sus conexiones — cuenta el total vs max_connections de la BD.

INTERMEDIO

GCP

Compute y contenedores

G088

RESPUESTA

Dataplane V2 (eBPF, Cilium): NetworkPolicies más eficientes, logging/observabilidad de flujos (kube API, flow export) — habilitarlo en la creación del cluster.

Sin DPv2: Calico alternativo.

Policy audit mode: loguea sin bloquear para calibrar antes de enforce.

Reglas ejemplo: frontend→backend:8080 only, backend→db:5432 only.

AVANZADO

GCP

Compute y contenedores

G087

RESPUESTA

K8s NetworkPolicy (Calico/Dataplane V2): default deny entre namespaces y permitir solo lo necesario (frontend→backend:8080, backend→db:5432).

Sin políticas, cualquier pod habla con todo (lateral movement fácil si uno se compromete).

Aplicar por fases: audit con logging primero, luego deny.

AVANZADO

GCP

Compute y contenedores

G090

RESPUESTA

NAP: crea node pools automáticamente según los pods pendientes (por tipo de máquina/GPU) — no precisas pools.

Combinar con: cluster autoscaler scale-down (vaciar nodos), pod anti-affinity razonable (no desperdiciar nodos), requests REALES de recursos (over-request = dinero tirado).

Ver nodos zombis: kubectl get nodes + utilization dashboard.

AVANZADO

GCP

Compute y contenedores

G089

RESPUESTA

StatefulSet: identidad estable (nombre ordinal), PVCs por réplica (volumeClaimTemplates) — para BDs autoalojadas.

PD CSI driver: persistent disks/zonal, Filestore para NFS, Cloud SQL proxy en vez de correr DB en k8s (recomendado).

Backup: Velero a GCS.

Regla de oro: en GKE el state EXTERNO (managed DBs), el clúster Stateless.

AVANZADO

GCP

★

Serverless

G091

Cloud Functions vs Cloud Run

BASICO

GCP

★

Serverless

G092

Cloud Run: revisiones y traffic split (canary fácil)

BASICO

GCP

★

Serverless

G093

Cloud Run: concurrencia y CPU allocation

BASICO

GCP

★

Serverless

G094

Min instances y cold starts en Cloud Run

BASICO

GCP

Serverless

G095

Cloud Run jobs y sidecars: capacidades menos conocidas

BASICO

GCP

Serverless

G096

Cloud Functions 2ª gen: qué hereda de Cloud Run

BASICO

GCP

Serverless

G097

Eventarc: el bus de eventos estándar de GCP

BASICO

GCP

Serverless

G098

App Engine en 2024+: ¿todavía tiene sentido?

BASICO

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

GCP ★
RESPUESTA

Serverless

G092

Cada deploy crea una REVISIÓN inmutable (imagen + env + recursos).
 gcloud run services update-traffic svc --to-revisions svc-00002=10,svc-00001=90 — canary sin Kubernetes.
 Tag una revisión (--tag blue) para URL de preview antes de servir tráfico.
 Rollback = redirigir 100% a la revisión anterior (instantáneo).

BASICO

GCP ★
RESPUESTA

Serverless

G091

Cloud Functions: FaaS — función suelta disparada por evento (HTTP, Pub/Sub, Storage); la 2ª generación corre sobre Cloud Run.
 Cloud Run: contenedor completo (cualquier runtime/framework), más control, alta concurrencia por instancia.
 Regla: una función pequeña → Functions; una app/servicio → Run.

BASICO

GCP ★
RESPUESTA

Serverless

G094

Frío: nueva instancia = pull imagen + arranque JVM (1-3s+).
 Min instances = N instancias siempre calientes (costo fijo pero latencia estable) — 1-2 para APIs críticas.
 Reducir frío: imagen distroless pequeña, JVM con CDS/AppCDS, arranque lazy de beans, startup CPU boost activado.

BASICO

GCP ★
RESPUESTA

Serverless

G093

Concurrency (default 80, max 1000): requests simultáneos por instancia — Node/async lo usa bien; Java/thread-per-request bajo (4-16) con más instancias.
 CPU allocation: solo-durante-request (barato, latencias frías al volver) vs always-allocated (background threads, gRPC servers, manteniendo cachés calientes).
 Ajuste = costo vs latencia.

BASICO

GCP
RESPUESTA

Serverless

G096

Gen2 ES Cloud Run por debajo: cualquier runtime con Functions Framework, timeouts hasta 60 min, más memoria/CPU, concurrency, Eventarc triggers nativos (60+ fuentes de eventos), tráfico por revisiones.
 Gen1: legado (timeouts 9 min, triggers limitados).
 Nuevo código: siempre gen2 o Cloud Run directo.

BASICO

GCP
RESPUESTA

Serverless

G095

Jobs: tareas batch one-shot o programadas (migraciones, ETL) con paralelismo por task index — no atado a HTTP.
 Sidecars (gen2): contenedores auxiliares en la misma instancia (proxy de service mesh, collector de logs) con lifecycle compartido.
 Ambos mantienen el modelo serverless sin Kubernetes.

BASICO

GCP
RESPUESTA

Serverless

G098

Standard: runtimes gestionados muy simples, scale to zero histórico, pero restricciones de runtime.
 Flexible: VMs gestionadas (menos serverless de verdad).
 Nuevos proyectos: Cloud Run cubre el 95% (contenedor portable, concurrencia, revisiones).
 App Engine sigue operado para apps existentes; no es la recomendación de partida hoy.

BASICO

GCP
RESPUESTA

Serverless

G097

Rutea eventos (Cloud Audit Logs, Pub/Sub, GCS, Firebase, custom) a Cloud Run/Functions/GKE con filtros por tipo/atributos.
 Ejemplo: 'cuando un objeto se suba a gs://facturas con prefijo xe/', invoca Cloud Run procesar.
 Reemplaza triggers ad-hoc de gen1; entrega at-least-once con retry configurable.

BASICO

GCP

★

Serverless

G099

Elegir ambiente de ejecución: matriz de decisión rápida

BASICO

GCP

Serverless

G100

Cloud Run y websockets/SSE: limitaciones reales

BASICO

GCP

Serverless

G101

Cold start de JVM en Cloud Run: checklist completo

BASICO

GCP

★

Almacenamiento

G102

Cloud Storage: clases y lifecycle

BASICO

GCP

Almacenamiento

G103

GCS: estructura (buckets/objetos) y límites clave

BASICO

GCP

★

Almacenamiento

G104

Signed URLs: dar acceso temporal sin credenciales

BASICO

GCP

★

Almacenamiento

G105

Lifecycle rules: automatizar costo y retención

BASICO

GCP

Almacenamiento

G106

Versioning, retention y bucket lock: inmutabilidad

BASICO

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

GCP

Serverless

G100

RESPUESTA

SSE/streaming HTTP: soportado bien (response streaming) — dashboards push simples.

WebSockets: soportados (gen2) PERO con concurrencia por instancia: cada instancia mantiene sus conexiones; scaling por conexiones requiere métricas custom.

Always-allocated CPU obligatorio para mantener conexiones vivas entre requests.

Timeouts: 60 min máx por request; reconexión del cliente debe estar diseñada.

BASICO

GCP



Serverless

G099

RESPUESTA

Contenedor HTTP/API → Cloud Run.

Función pequeña evento-driven → Cloud Functions gen2.

Tarea batch → Cloud Run Jobs (o Batch service para HPC).

Muchos microservicios con malla/ecosistema k8s → GKE.

VM con software legado/licencias → Compute Engine.

Sin servidores y con BD gestionada → App Engine solo si ya vives ahí.

BASICO

GCP



Almacenamiento

G102

RESPUESTA

Standard: acceso frecuente.

Nearline: ~1 acceso/mes. Coldline: ~1 cada 90 días. Archive: retención larga (lo más barato).

Multi-region para HA; region para menor costo.

Lifecycle rules automáticas: degradar clase, borrar versiones viejas. Versioning y retención.

BASICO

GCP

Serverless

G101

RESPUESTA

1) Startup CPU boost ON (2 vCPU al arrancar).

2) Imagen distroless + JVM headless; AppCDS/CDS training para compartir clases.

3) Spring: lazy initialization selectivo, excluir autoconfigs de arranque, iniciar listeners tras ready (readiness).

4) Min instances = 1 para rutas críticas.

5) Medir: container startup time en métricas de Run — objetivo <2s con boost.

BASICO

GCP



Almacenamiento

G104

RESPUESTA

URL firmada con expiración (V4): permite al cliente SUBIR o DESCARGAR directo del bucket con tu firma — tu app solo genera la URL.

Casos: uploads de archivos grandes sin atravesar tu API, links de descarga seguros temporales.

Signed POST policies para restringir tamaño/CONTENT-type en upload.

BASICO

GCP

Almacenamiento

G103

RESPUESTA

Namespace global de buckets; dentro, objetos planos con nombre (no hay 'carpetas' reales — son prefijos).

Objetos inmutables por versión; hasta 5TB por objeto; metadata + ACLs/IAM.

Ops por HTTP/XML/JSON API, gsutil o client libs con resumable upload para grandes.

Diseño de nombres: prefijos tipo fecha (logs/2025/09/12/) para listar/particionar.

BASICO

GCP

Almacenamiento

G106

RESPUESTA

Versioning: sobrescribir crea versión antigua accesible (restore/soft-delete accidental).

Retention policy: objetos no borrables antes de N días (compliance). Bucket Lock con retención LOCKED = WORM immutable (auditoría legal).

Soft delete (nuevo): ventana de recuperación de objetos borrados por defecto (7 días).

BASICO

GCP



Almacenamiento

G105

RESPUESTA

Reglas por prefijo/edad/tiempo desde última modificación: cambiar a Nearline (30d), Coldline (90d), Archive (365d), borrar versiones viejas, eliminar objetos temporales.

Ejemplo: logs → 90 días Standard, luego Archive, borrar a 3 años (compliance).

Se aplican de fondo; verificar con lifecycle log/dry runs.

BASICO

GCP

Almacenamiento

G107

Public objects y uniform access: seguridad de buckets

BASICO

GCP

Almacenamiento

G108

Storage Transfer Service: migraciones y sincronización

BASICO

GCP

Almacenamiento

G109

Sitio estático + CDN en GCS: la arquitectura más barata

BASICO

GCP

Almacenamiento

G110

Diseño de buckets multi-entorno sin filtros de datos cruzados

BASICO

GCP

Bases de datos

G111

¿Para qué sirve Memorystore?

BASICO

GCP

★

Bases de datos

G112

Cloud SQL: HA, réplicas y backups de verdad

INTERMEDIO

GCP

★

Bases de datos

G113

Conectarte a Cloud SQL: proxy vs IP privada

INTERMEDIO

GCP

Bases de datos

G114

Cloud SQL vs AlloyDB: cuando Postgres crece

INTERMEDIO

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

GCP

Almacenamiento

G108

RESPUESTA

Transfiere: de S3/Azure/on-prem a GCS, entre buckets, o respaldo recurrente con filtros (prefijo, fecha) y schedules.
Verificación de integridad (checksums), bandwidth limit, delete objects que no estén en el origen (sync).
Transfer Appliance: hardware físico para TB/PB sin red suficiente.

BASICO

GCP

Almacenamiento

G107

RESPUESTA

Uniform bucket-level access (obligatorio en nuevos): SOLO IAM, sin ACLs por objeto — gobernanza simple.
allUsers/allAuthenticatedDomains = público: detectar con SCC y bloquear con org policy (publicAccessPrevention).
Compartir: signed URLs o IAM por grupo, nunca ACLs ad-hoc.

BASICO

GCP

Almacenamiento

G110

RESPUESTA

Separación por proyecto (dev/stage/prod) — cada entorno su bucket, su IAM.
Dentro: prefijos por dominio y fecha (invoices/2025/09/). Evitar bucket único 'todo' con ACLs finas.
Permisos por prefijo solo si IAM uniforme está OFF (anti-patrón) — mejor buckets separados.
Nombre del bucket = identidad global: usar sufijo random/hashed para evitar colisiones.

BASICO

GCP

Almacenamiento

G109

RESPUESTA

Bucket con index.html + LB backend bucket + Cloud CDN → sitio global HTTPS con cert gestionado.
Cache-Control por tipo (html no-cache, assets 1 año immutable con hash en nombre).
Alternativa serverless: Firebase Hosting (deploy en 1 comando).
Costo: centavos para tráfico moderado.

BASICO

GCP



Bases de datos

G112

RESPUESTA

HA: regional con standby sincrónico en otra zona — failover automático (~60s, misma IP via proxy).
Read replicas: lecturas escaladas / analítica ligera; cross-region replica para DR (promotable).
Backups automáticos + PITR con write-ahead logs (restaurar a un minuto concreto).
Maintenance window configurable (patches).

INTERMEDIO

GCP

Bases de datos

G111

RESPUESTA

Redis / Memcached gestionado en memoria:

- caché de resultados y sesiones
- rate limiting
- colas ligeras

Reduce latencia (sub-milisegundos) y carga sobre Cloud SQL/Firestore.
Tiers: Basic (sin HA) vs Standard (con réplica). Configurar eviction (allkeys-lru) y TTLs.

BASICO

GCP

Bases de datos

G114

RESPUESTA

Cloud SQL: Postgres/MySQL gestionado clásico — HA, réplicas, precio contenido.
AlloyDB: Postgres compatible pero con capa analítica (columnar engine 100x para queries analíticas), ~2x más rápido OLTP, escáner de caché ultra rápido, escala de réplicas de lectura sin lag.
Candidato cuando Cloud SQL se queda corto en performance y el presupuesto existe.

INTERMEDIO

GCP



Bases de datos

G113

RESPUESTA

Cloud SQL Auth Proxy: sidecar/proceso local que cifra y autentica por IAM (sin SSL certs manuales ni IP pública) — recomendado para Cloud Run/develop.
Private IP: la BD solo en la VPC (Serverless VPC connector / GKE nativo) — topología segura de prod.
Jamás IP pública + password solo (combinable pero mínima).

INTERMEDIO

GCP

Bases de datos

G115

Firestore: modelo de datos y reglas

INTERMEDIO

GCP

Bases de datos

G116

Firestore: límites que definen si te sirve

INTERMEDIO

GCP

★

Bases de datos

G117

Matriz de decisión de bases de datos GCP (entrevista clásica)

INTERMEDIO

GCP

Bases de datos

G118

Memorystore para Redis: HA, tamaño y patrones de uso

INTERMEDIO

GCP

★

Bases de datos

G119

Caching pattern: cache-aside implementado bien

INTERMEDIO

GCP

Bases de datos

G120

Cloud SQL performance: los 6 knobs que mueven la aguja

INTERMEDIO

GCP

Bases de datos

G121

Secrets y datos sensibles en BDs: cifrado y acceso

INTERMEDIO

GCP

Bases de datos

G122

Cloud SQL flags y mantenimiento: los detalles que evitan sustos

INTERMEDIO

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

GCP

Bases de datos

G116

RESPUESTA

1MB/doc, 500 writes/seg por documento (hotspot), sin queries OR amplios ni 'NOT IN' complejos, consistencia eventual entre documentos (tx solo sobre doc/grupo).

Diseño: denormalizar, arrays acotados (array de 10k = antipatrón), distribuir writes con sharded counters.

Analítica pesada → exportar a BigQuery.

INTERMEDIO

GCP

Bases de datos

G115

RESPUESTA

Documentos JSON (~1MB) en colecciones; subcolecciones anidadas; documentos con IDs.

Consultas con índices (compuestos automáticos/manuales); SIN joins ni agregados complejos (count/sum básicos).

Realtime listeners y offline sync (móvil/web) — su superpoder.

Security Rules: permisos por documento evaluados en el cliente-borde (para apps directas).

INTERMEDIO

GCP

Bases de datos

G118

RESPUESTA

Tiers: Basic (sin réplica, barato) vs Standard (réplica + failover); cluster mode para >5GB o sharding.

Uso: caché de resultados, sesiones, rate limiting (INCR), leaderboards (sorted sets), locks con SET NX PX.

Eviction policy: allkeys-lru típico de caché; volatile-ttl si mezclas.

Persistencia (RDB) opcional — Redis como caché no la necesita.

INTERMEDIO

GCP



Bases de datos

G117

RESPUESTA

Relacional simple gestionado → Cloud SQL. Postgres a escala/analítica → AlloyDB. Relacional global ultra-consistente → Spanner. Documentos + realtime/offline móvil → Firestore. Series temporales/IoT masivo → Bigtable. Caché/sesiones → Memorystore. Analítica OLAP → BigQuery. Blobs/archivos → Cloud Storage.

Pregunta clave: patrón de acceso + consistencia + escala + presupuesto.

INTERMEDIO

GCP

Bases de datos

G120

RESPUESTA

- 1) Tamaño de instancia correcto (CPU/RAM p95).
- 2) Índices con EXPLAIN ANALYZE (missing index = sec scan).
- 3) Connection pooling: Hikari + max_connections coherentes; PgBouncer si apps múltiples.
- 4) flags: innodb_buffer_pool_size / shared_buffers ~70% RAM.
- 5) read replica para analítica.
- 6) SSD PD y maintenance window fuera de picos.

INTERMEDIO

GCP



Bases de datos

G119

RESPUESTA

Lectura: 1) buscas en Redis, 2) miss → BD, 3) guardas en Redis con TTL + jitter.

Escritura: actualiza BD → invalida/actualiza caché (delete, no write-through, para evitar race).

Claves con namespaces: prod:user:42.

Serialize DTOs compactos (JSON/protobuf); nunca entidades JPA con lazy.

INTERMEDIO

GCP

Bases de datos

G122

RESPUESTA

Flags (database flags): slow_query_log=ON, max_connections, timezone, character_set.

Maintenance window (día/hora) + maintenance timing (earlier/later) — Google patchea ahí; HA hace failover real (prueba el rebalanceo).

Rejections en producción: siempre el warning de 'patching imminent' por email/monitoring.

Instance restart por flag = planificar ventana.

INTERMEDIO

GCP

Bases de datos

G121

RESPUESTA

At-rest: cifrado por defecto; CMEK si compliance exige.

En tránsito: SSL obligatorio (force SSL flag) + IAM-auth (Cloud SQL IAM authentication: usuarios = identidades Google).

Passwords de app: Secret Manager, nunca en properties.

Auditoría: Cloud SQL audit logs para SELECT/UPDATE de tablas sensibles (PII).

INTERMEDIO

GCP

★

Bases de datos

G123

¿Qué base de datos GCP elijo? (SQL vs Firestore vs Spanner vs Bigtable vs BigQuery)

AVANZADO

GCP

Bases de datos

G124

Spanner: qué lo hace único (TrueTime y global)

AVANZADO

GCP

Bases de datos

G125

Bigtable: diseño de row key que todo el mundo hace mal

AVANZADO

GCP

Bases de datos

G126

Database Migration Service: migrar con mínimo downtime

AVANZADO

GCP

Bases de datos

G127

Cross-region disaster recovery para Cloud SQL: pasos

AVANZADO

GCP

Bases de datos

G128

Vector databases en GCP: opciones actuales

AVANZADO

GCP

★

Mensajería y eventos

G129

Pub/Sub: modelo y garantías de entrega

INTERMEDIO

GCP

Mensajería y eventos

G130

Pub/Sub vs Kafka vs RabbitMQ: mapa conceptual

INTERMEDIO

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

GCP

Bases de datos

G124

RESPUESTA

SQL relacional DISTRIBUIDO con consistencia EXTERNA (strong) global — TrueTime (relojes atómicos GPS) permite transacciones serializables a escala planetaria.
Escalas horizontalmente sin sharding manual (interleave/tablas hijas para localidad).
Costo alto (nodos + storage): para fin-tech, inventario global, SaaS multi-tenant serio.
99.999% multi-región.

AVANZADO

GCP



Bases de datos

G123

RESPUESTA

Cloud SQL: relacional gestionado (MySQL/Postgres/SQL Server) — apps transaccionales.
Firestore: NoSQL documentos, tiempo real (móvil/web).
Spanner: SQL distribuido GLOBAL con consistencia fuerte.
Bigtable: NoSQL wide-column, time-series/IoT, throughput masivo.
BigQuery: analítica OLAP sobre petabytes (serverless).

AVANZADO

GCP

Bases de datos

G126

RESPUESTA

DMS: réplica continua desde MySQL/Postgres/SQL Server (on-prem u otro cloud) a Cloud SQL/AlloyDB con CDC (binlog/WAL).
Cutover: cortar escrituras origen → esperar lag 0 → promover destino → redirigir app.
Homogeneous principalmente; heterogéneo vía tools adicionales (PgBouncer/assessment).
Prueba: run pre-migration assessment.

AVANZADO

GCP

Bases de datos

G125

RESPUESTA

Rendimiento = distribución de filas por row key. MAL: timestamp puro al inicio (todo cae en un nodo — hotspot). BIEN: prefijo reversado/distribuido + timestamp reversado para 'lo más nuevo primero': user#reversed_ts.
Wide-column: miles de versiones por celda, GC policies por edad/versión.
Casos: series temporales IoT, telemetría, fin personalizado de HBase.

AVANZADO

GCP

Bases de datos

G128

RESPUESTA

1) Vertex AI Vector Search (Managed Index, ANN a escala — el más potente).
2) AlloyDB con extensiones pgvector + fast path — Postgres friendly.
3) Cloud SQL pgvector para escala moderada.
4) Firestore vector search para apps móviles.
Patrón RAG: documentos → embeddings (Vertex) → índice → query embedding → top-k → LLM con contexto.

AVANZADO

GCP

Bases de datos

G127

RESPUESTA

1) Read replica cross-region continua (asincrónica) en la región DR.
2) Backups también en la región secundaria (backup replication).
3) Drill: promover la réplica (promote) y apuntar la app vía connection name por región/flag.
4) RTO estimado = promoción (~min) + redirect; RPO = lag de réplica (segundos-min).
Automatizar el failover con Workflows + alerta de región caída.

AVANZADO

GCP

Mensajería y eventos

G130

RESPUESTA

Pub/Sub: managed global, push/pull, at-least-once, sin claves de partición complejas — el default GCP.
Kafka: log distribuido con particiones y consumer groups, replay nativo, exactly-once con transacciones — streaming pesado y ecosistema (self-managed o Dataproc/Confluent).
RabbitMQ: colas inteligentes con routing (exchanges) — patrones de cola enterprise clásicos.

INTERMEDIO

GCP



Mensajería y eventos

G129

RESPUESTA

Publisher → topic → subscription → subscriber (pull o push).
Entrega AT-LEAST-ONCE: puede duplicar → el consumidor debe ser idempotente.
Ack deadline: si no confirmas, se reenvía.
Dead-letter topic tras N intentos fallidos. Ordering keys para orden por clave.

INTERMEDIO

GCP

★

Mensajería y eventos

G131

Push vs Pull en Pub/Sub: cuándo cada uno

INTERMEDIO

GCP

Mensajería y eventos

G132

Pub/Sub schemas y filtrado de subscriptions

INTERMEDIO

GCP

★

Mensajería y eventos

G133

Cloud Tasks vs Pub/Sub: la pregunta que confunde a todos

INTERMEDIO

GCP

Mensajería y eventos

G134

Event-driven en GCP: la arquitectura canónica

INTERMEDIO

GCP

Mensajería y eventos

G135

Cloud Scheduler + Workflows: orquestación sin servidores

INTERMEDIO

GCP

Mensajería y eventos

G136

Idempotencia de consumidores: implementación concreta

INTERMEDIO

GCP

Mensajería y eventos

G137

Poison messages y DLQ handling operativo

INTERMEDIO

GCP

Mensajería y eventos

G138

Ack deadline, max delivery y dead letter: el ciclo de vida del mensaje

AVANZADO

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

GCP

Mensajería y eventos

G132

RESPUESTA

Schema registry (Avro/Protobuf): valida el payload al publicar — contrato versionado con compatibilidad (backward/forward).
 Filter de subscription: atributos (type: "order.created" && region: "eu") → cada servicio solo recibe lo suyo sin fan-out manual.
 BigQuery subscription: escribe directo a tabla (ETL sin consumer).
 Retention de mensajes (hasta 31 días) para replay.

INTERMEDIO

GCP



Mensajería y eventos

G131

RESPUESTA

Pull: el consumidor pide mensajes (control total de ritmo, ideal para workers y batch).
 Push: Pub/Sub llama a tu endpoint HTTPS (webhooks, Cloud Run) — simple, escala automática, pero control limitado y el receptor debe responder 2xx rápido.
 Push con OIDC token para autenticar el caller.
 Streaming pull: alta tasa con cliente gestionando más eficientemente.

INTERMEDIO

GCP

Mensajería y eventos

G134

RESPUESTA

Origen: app cambia estado → publica a Pub/Sub (o Eventarc escucha Audit Logs de GCS/BQ).
 Eventos fluyen: subscriptions filtradas por tipo → Cloud Run consumers (escalado por mensaje).
 Efectos: BD actualizada, notificaciones, analítica (BigQuery subscription).
 Reglas: eventos inmutables en pasado ('PedidoCreado'), payload con id + datos mínimos o claim-check.

INTERMEDIO

GCP



Mensajería y eventos

G133

RESPUESTA

Pub/Sub: eventos FAN-OUT a muchos suscriptores, throughput masivo, sin control fino de timing.
 Cloud Tasks: comandos punto a punto UNO A UNO con control fino: tasa de dispatch (rate limits), schedule en el futuro, retry backoff explícito, token OIDC.
 Casos Tasks: emails con rate limit de proveedor, callbacks programados, throttling a API externa.

INTERMEDIO

GCP

Mensajería y eventos

G136

RESPUESTA

Clave = eventId/messageId del evento.
 Tabla dedup: INSERT INTO processed(event_id) — si constraint violation, ya procesado → ack y skip (en la MISMA tx del negocio = atómico).
 TTL/purga de la tabla. Redis SETNX con expiración como caché de dedup rápida.
 Operaciones naturales idempotentes: upsert, set estado por versión.

INTERMEDIO

GCP

Mensajería y eventos

G135

RESPUESTA

Scheduler: cron gestionado que llama HTTP/PubSub/AppEngine (UI de cron, timezone).
 Workflows: orquestador YAML/JSON con pasos, condiciones, retries, llamadas a APIs — el 'paso a paso' serverless (sustituye scripts glue).
 Ejemplo: Scheduler diario → Workflow: extraer datos, llamar Cloud Run, si falla retry, notificar.
 Barato y sin infraestructura.

INTERMEDIO

GCP

Mensajería y eventos

G138

RESPUESTA

Mensaje entregado queda 'in flight' hasta ack o expiración del ackDeadline (10-600s) → modAck para extender en procesamiento largo.
 Tras N intentos (delivery attempt/dead letter policy en la subscription) → topic dead-letter con header de causa.
 Consumidor idempotente OBLIGATORIO (at-least-once).
 Seek permite re-entregar (snapshot/fecha) para re-procesar.

AVANZADO

GCP

Mensajería y eventos

G137

RESPUESTA

Mensaje que SIEMPRE falla (payload inválido): dead-letter policy tras N intentos evita bloquear la cola.
 Operación: monitorear profundidad del DLQ (alerta), inspeccionar causa (header stack/reason), fix del bug, re-publicar corregidos (seek o republish tool).
 Sin DLQ: un solo mensaje malo puede parar todo el pipeline.

INTERMEDIO

GCP

Mensajería y eventos

G139

Ordering keys: orden garantizado y su costo

AVANZADO

GCP

Mensajería y eventos

G140

Pub/Sub Lite vs Pub/Sub: el caso especial

AVANZADO

GCP

Mensajería y eventos

G141

Ordering + exactly-once: recipe final con Pub/Sub

AVANZADO

GCP

Analítica de datos

G142

BigQuery: modelo de costos on-demand vs slots

INTERMEDIO

GCP

★

Analítica de datos

G143

Partitioning y clustering en BigQuery: el 80% de la performance

INTERMEDIO

GCP

Analítica de datos

G144

Cargar datos a BigQuery: las 5 rutas

INTERMEDIO

GCP

Analítica de datos

G145

Looker Studio / Looker: el extremo BI

INTERMEDIO

GCP

Analítica de datos

G146

Dataflow (Apache Beam): batch y streaming unificados

AVANZADO

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

GCP

Mensajería y eventos

G140

RESPUESTA

Lite: particiones/zonas fijas, costo por capacidad reservada MUCHO menor para streaming continuo masivo (telemetría), modelo Kafka-like (offsets).
 Pub/Sub: global, dinámico, features ricas (schemas, filters, push).
 Lite recomendado para volúmenes predecibles enormes con presupuesto apretado; Pub/Sub para todo lo demás.
 (Nota: está en sunset gradual hacia managed Kafka — evaluar BigQuery subscription/direct Kafka).

AVANZADO

GCP

Mensajería y eventos

G139

RESPUESTA

messageOrdering=true en publisher + orderingKey (ej. orderId) → mensajes de la MISMA clave llegan en orden al mismo suscriptor.
 Costo: throughput reducido (una clave = una secuencia); si un mensaje falla, se BLOQUEA la clave hasta ack/expiración (entrega la detiene).
 Exactly-once delivery (región con feature) refuerza orden + no duplicados.
 Alternativa: procesar orden en el consumidor por versiones/sequence.

AVANZADO

GCP

Analítica de datos

G142

RESPUESTA

On-demand: pagas por bytes ESCANEADOS (~\$6/TB) — el WHERE por columna (columnar) importa; particionar reduce el scan.
 Editions/slots: capacidad reservada (slots por segundo/hora/mensual) — predecible y mejor a escala (sin límites de bytes).
 Controles: custom quotas por usuario/proyecto, maximum bytes billed por query, dry run antes de correr.

INTERMEDIO

GCP

Mensajería y eventos

G141

RESPUESTA

Ordering key por entidad + enable_message_ordering en subscription + endpoint regional (mismo región del topic).
 Exactly-once delivery (regiones soportadas): no duplicados con retry del mismo mensaje.
 Aun así: consumidor idempotente (dedup table) — la defensa final.
 Monitorear 'oldest unacked message' por ordering key para bloqueos.

AVANZADO

GCP

Analítica de datos

G144

RESPUESTA

- 1) Batch load (CSV/Parquet/AVRO desde GCS) — gratis, ideal para cargas grandes puntuales.
- 2) Streaming insert (write API) — latencia segundos, costo por GB.
- 3) Storage Write API (nuevo, exactly-once, transactions) — el estándar para pipelines.
- 4) BigQuery subscription desde Pub/Sub — directo desde mensajería.
- 5) Federated queries (tabla externa sobre GCS/Cloud SQL) — sin mover datos.

INTERMEDIO

GCP



Analítica de datos

G143

RESPUESTA

Partition por DATE/TIMESTAMP (o integer range): cada query con WHERE fecha = 'X' escanea UNA partición (no toda la tabla).
 Clustering por columnas frecuentes de filtro (user_id, pais): ordena bloques dentro de la partición → skips.
 Result: terabytes evitados = dinero y velocidad.
 Re-particionar tabla grande: create table partition by... as select.

INTERMEDIO

GCP

Analítica de datos

G146

RESPUESTA

Pipeline en Java/Python → corre en managed Dataflow: auto-scaling de workers, exactly-once en streaming.
 Casos: ETL por lotes, procesamiento de streams (ventanas, agregaciones), correcciones tardías (watermarks).
 Costo por vCPU/RAM de workers — tuning con autoscaling y streaming engine.
 Alternativa light: Dataproc (Spark) si el equipo ya tiene Spark.

AVANZADO

GCP

Analítica de datos

G145

RESPUESTA

Looker Studio (gratis): dashboards conectados a BigQuery/Sheets — reportes operativos rápidos.
 Looker (enterprise): capa semántica (LookML) — métricas definidas UNA vez, gobernadas, self-service real.
 Patrón de entrega de datos: BigQuery mart → dashboard ejecutivo.
 Embed: dashboards embebidos para clientes SaaS.

INTERMEDIO

GCP

Analítica de datos

G147

Lakehouse en GCP: GCS + BigQuery + Dataplex

AVANZADO

GCP

Analítica de datos

G148

Composer (Airflow gestionado): DAGs como código

AVANZADO

GCP

Analítica de datos

G149

Streaming pipeline canónico: Pub/Sub → Dataflow → BigQuery

AVANZADO

GCP

Analítica de datos

G150

BigQuery ML: modelos sin salir del almacén

AVANZADO

GCP

Analítica de datos

G151

Vertex AI: qué ofrece end-to-end

AVANZADO

GCP

★

DevOps y CI/CD

G152

CI/CD en GCP: Cloud Build + Artifact Registry

INTERMEDIO

GCP

★

DevOps y CI/CD

G153

Cloud Build: pipeline básico en cloudbuild.yaml

INTERMEDIO

GCP

★

DevOps y CI/CD

G154

Triggers de Cloud Build: por branch, tag, PR y manual

INTERMEDIO

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

GCP

Analítica de datos

G148

RESPUESTA

Composer = Apache Airflow gestionado: DAGs en Python versionados en GCS, schedule/orquestación con dependencias entre tareas.
Casos: pipelines ETL multi-step (extraer Cloud SQL → Dataflow → BigQuery → Looker), ML retraining.
Operación: entorno en su propio GKE (costo fijo mensual notable), monitoring de tasks fallidas.
Alternativa serverless: Workflows para orquestaciones simples.

AVANZADO

GCP

Analítica de datos

G147

RESPUESTA

Data lake: GCS (Parquet) como almacenamiento barato por zonas (raw/curated/consumed).
BigQuery consulta lakehouse: tablas externas / BigLake (metadatos, seguridad a nivel fila/columna, caché).
Dataplex: catálogo, calidad y gobernanza por zonas.
ETL con Dataflow/Datapro; BI con Looker/Looker Studio.

AVANZADO

GCP

Analítica de datos

G150

RESPUESTA

CREATE MODEL ... OPTIONS(model_type='logistic_reg') AS SELECT ... — entrena en SQL.
Predicción: ML.PREDICT(MODEL m, TABLE t); evaluación MLEVALUATE.
Tipos: regresión, clasificación, clustering (kmeans), time series (ARIMA), matrices.
Casos: churn simple, forecasting de demanda — sin mover datos a otra plataforma.
Para ML serio: Vertex AI, pero BQML cubre el 80% analítico.

AVANZADO

GCP

Analítica de datos

G149

RESPUESTA

Eventos (clicks, sensores, transacciones) → Pub/Sub topic → Dataflow streaming: parse/validate/enriquecer, ventanas por minuto → BigQuery tablas particionadas.
Exactly-once con Storage Write API sink; late data con watermarks.
Monitoreo: system lag de Pub/Sub, watermarks de Dataflow, frescura de BQ.
Misma topología sirve para agregados en tiempo real (Firestore/Memorystore sink).

AVANZADO

GCP ★

DevOps y CI/CD

G152

RESPUESTA

Cloud Build: ejecuta pipelines definidos en cloudbuild.yaml (build, test, push, deploy) con triggers desde Git.
Artifact Registry: almacén de imágenes Docker y paquetes (npm, Maven, Python) con IAM y análisis de vulnerabilidades.
Flujo típico: build → push imagen → gcloud run deploy.

INTERMEDIO

GCP

Analítica de datos

G151

RESPUESTA

Datasets + entrenamiento (AutoML / custom containers), Feature Store, Model Registry, endpoints de serving (online/batch), Pipelines (Kubeflow) para MLOps, model monitoring de drift.
Generative: Vertex AI Studio, Gemini API, grounding con tus datos (RAG), embeddings + Vector Search.
Costo: cómputo por hora de entrenamiento + serving por nodo-hora.
Es el 'AWS SageMaker' de GCP.

AVANZADO

GCP ★

DevOps y CI/CD

G154

RESPUESTA

Push a branch (main→deploy stage), push a tag (v*→prod), Pull Request (solo build/test), manual (deploy one-off).
Filtros por ruta incluidos/excluded files (solo build si cambia /api).
Service account del build con permisos mínimos (deploy a Run, nada más).
Aprobaciones (approvals) antes de desplegar a prod.

INTERMEDIO

GCP ★

DevOps y CI/CD

G153

RESPUESTA

steps: [{name: gcr.io/cloud-builders/docker, args: [build, -t, \$LOCATION-docker.pkg.dev/\$PROJECT_ID/repo/app:\$SHORT_SHA, .]} , {name: docker, args: [push, ...]}]
images, substitutions (\$PROJECT_ID, \$SHORT_SHA, \$_VARS custom), availableSecrets para tokens.
gcloud builds submit --config cloudbuild.yaml .
Trigger desde GitHub: push a main → build → deploy.

INTERMEDIO

GCP

★

DevOps y CI/CD

G155

Secretos en Cloud Build (sin exponerlos en logs)

INTERMEDIO

GCP

DevOps y CI/CD

G156

CD a Cloud Run: build → deploy → smoke test

INTERMEDIO

GCP

DevOps y CI/CD

G157

Estrategia de branching y deploy por entornos en GCP

INTERMEDIO

GCP

DevOps y CI/CD

G158

Terraform en GCP: estructura de proyecto típica

INTERMEDIO

GCP

DevOps y CI/CD

G159

Cloud Deploy: progresión gestionada dev→prod

INTERMEDIO

GCP

DevOps y CI/CD

G160

Pipelines de imágenes seguras: scan → sign → deploy

INTERMEDIO

GCP

DevOps y CI/CD

G161

Feature flags en pipeline de GCP: decoupling deploy/release

INTERMEDIO

GCP

DevOps y CI/CD

G162

Monorepo vs multirepo para microservicios + CI

INTERMEDIO

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

GCP

DevOps y CI/CD

G156

RESPUESTA

Step 1 docker build/push; step 2 gcloud run deploy svc --image ... --region --quiet; step 3 curl smoke test al URL de la revisión tagged; si falla → rollback traffic a revisión anterior.
 Alternativa declarativa: Cloud Deploy (pipeline dev→stage→prod con approvals y rollbacks unificados para Run/GKE).
 Todo como código en el repo del servicio.

INTERMEDIO

GCP



DevOps y CI/CD

G155

RESPUESTA

availableSecrets: secretManager: [versionName: projects/p/secrets/npm-token/versions/latest] + step env: NPM_TOKEN=secretManager:npm-token.
 El valor no aparece en logs (enmascarado).
 Credenciales de deploy: la SA del build con roles mínimos, impersonation en lugar de keys.
 Nunca COPY secrets en la imagen.

INTERMEDIO

GCP

DevOps y CI/CD

G158

RESPUESTA

modules: network (VPC/subnets/NAT), project-factory (proyecto + APIs + budgets), run-service (deploy app), data (SQL + backup).
 Environments: dev.tfvars / prod.tfvars sobre el mismo módulo.
 Remote state en GCS + lock. CI: plan en PR (comentario del plan), apply manual o auto en merge para dev.
 Version pinning de providers.

INTERMEDIO

GCP

DevOps y CI/CD

G157

RESPUESTA

Trunk-based + feature flags: main siempre desplegable → dev automático; stage con tag/merge; prod con release tag + aprobación.
 Entornos = proyectos GCP distintos (misma receta Terraform) — nunca carpetas en el mismo proyecto.
 Promoción de la MISMA imagen entre entornos (solo cambia config) — no re-builds por entorno.

INTERMEDIO

GCP

DevOps y CI/CD

G160

RESPUESTA

Cloud Build → Artifact Registry con on-push vulnerability scanning (Critical/High bloquea) → attestación firmada (Binary Authorization) → solo imágenes firmadas despliegan en GKE/Run.
 SBOM generado (syft) adjunto; provenance SLSA de la build.
 Base images minimal (distroless/chainguard) reducen CVEs de raíz.

INTERMEDIO

GCP

DevOps y CI/CD

G159

RESPUESTA

Pipeline declarativo: targets (dev, stage, prod), strategy (standard o canary con % por fases), approvals en prod, rollbacks de 1 click.
 Renderiza scaffold profiles (config por target) y despliega a Run/GKE.
 Historial de releases y estados centralizado — para equipos sin plataforma propia de CD.

INTERMEDIO

GCP

DevOps y CI/CD

G162

RESPUESTA

Monorepo: atomicidad de cambios cruzados, refactors globales; CI con filtros por carpeta (solo build lo tocado); governance única.
 Multirepo: ownership y releases independientes, permisos simples; versiones y sync manual entre repos.
 GCP: Cloud Build triggers con included files para monorepo; un trigger por servicio en multirepo.
 Ambos válidos; el equipo manda.

INTERMEDIO

GCP

DevOps y CI/CD

G161

RESPUESTA

Deploy frecuente + flag apagado = código en prod invisible al usuario.
 Gestión: Unleash self-hosted en GKE / LaunchDarkly / Firebase Remote Config (móvil) / properties en Firestore.
 Kill switch para apagar feature problemática sin rollback.
 Higiene: dueño + fecha de expiración por flag (los flags eternos son deuda).

INTERMEDIO

GCP

DevOps y CI/CD

G163

Migraciones de schema seguras en CI/CD
(expand-contract)

INTERMEDIO

GCP

DevOps y CI/CD

G164

GitOps con GKE: ArgoCD/Config Sync

AVANZADO

GCP

★

DevOps y CI/CD

G165

Private pools de Cloud Build: builds en tu VPC

AVANZADO

GCP

★

Observabilidad y SRE

G166

Cloud Operations: Logging, Monitoring, Trace

INTERMEDIO

GCP

Observabilidad y SRE

G167

Cloud Logging: consultas y sinks que debes dominar

INTERMEDIO

GCP

Observabilidad y SRE

G168

Cloud Monitoring: uptime, alertas y políticas

INTERMEDIO

GCP

Observabilidad y SRE

G169

Cloud Trace y Profiler: diagnóstico de latencia y CPU

INTERMEDIO

GCP

Observabilidad y SRE

G170

Error Reporting: el dashboard de excepciones

INTERMEDIO

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

GCP

DevOps y CI/CD

G164

RESPUESTA

Git = fuente de verdad del estado del clúster: ArgoCD sincroniza manifiestos/Helm desde repo, diff visible, rollback = git revert.
 Config Sync (Anthos): sincroniza configs/namespaces/policies a nivel org.
 Separación: repo de app (código) vs repo de despliegue (manifiestos) — CI actualiza el segundo, CD sincroniza.
 Beneficio: auditoría y recuperación total del clúster desde git.

AVANZADO

GCP

DevOps y CI/CD

G163

RESPUESTA

Deploy N: migración ADDITIVA (añadir columna nullable + backfill) — compatible con versión vieja.
 Deploy código N (usa ambas). Deploy código N+1 (solo nueva).
 Deploy N+2: migración contractiva (drop viejo).
 Regla: cada migración debe correr contra el código de la versión PREVIA — nunca renombrar/dropear en el mismo release.
 Flyway/Liquibase versionados por servicio.

INTERMEDIO

GCP ★

Observabilidad y SRE

G166

RESPUESTA

Logging: logs centralizados de todos los servicios, consultas, sinks a BigQuery/Storage, log-based metrics.
 Monitoring: dashboards, uptime checks, alert policies y SLOs.
 Cloud Trace: distributed tracing de latencias entre servicios.
 Profiler: uso continuo de CPU/memoria.

INTERMEDIO

GCP ★

DevOps y CI/CD

G165

RESPUESTA

Worker pool dedicado dentro de tu VPC: acceso a recursos privados (Cloud SQL de test, Memorystore, repos internos) sin exponerlos.
 Control de máquina (tipo, disco) y de egress.
 Costo por tiempo de worker; usar solo para builds que requieren red privada — el resto en el pool compartido.

AVANZADO

GCP

Observabilidad y SRE

G168

RESPUESTA

Uptime checks: sonda externa cada N min desde varias regiones (URL, grupos).
 Alert policies: condición sobre métrica (error rate > 5% en 5m) → notificación (email, SMS, Slack, Pub/Sub, PagerDuty).
 Throttling/aggregation: ventanas, alineadores (mean, count), 'for X minutos' para evitar ruido.
 Incidentes con timeline y ack.

INTERMEDIO

GCP

Observabilidad y SRE

G167

RESPUESTA

Queries: resource.type="cloud_run_revision" severity>=ERROR
 textPayload:"timeout"; guardar queries.
 Sinks: enrutamiento a BigQuery (analítica), GCS (archivo barato), Pub/Sub (alertas custom/SIEM).
 Exclusion sinks: bajar costo descartando logs ruidosos en el origen.
 Log-based metrics: contar ocurrencias de un patrón → métrica + alerta.

INTERMEDIO

GCP

Observabilidad y SRE

G170

RESPUESTA

Agrupar stack traces idénticos de logs (Cloud Run/GKE/Functions escriben con formato estándar) → conteo, usuarios afectados, primera/última vez.
 Alerta sobre nuevo error group (fade-in).
 Para Java: loggear Throwable con severity=ERROR y el agente lo agrupa.
 Rutina: revisar grupos nuevos en cada deploy.

INTERMEDIO

GCP

Observabilidad y SRE

G169

RESPUESTA

Trace: waterfall de una request entre servicios (spans por llamada/SQL) — localizar el 300ms escondido; sampling configurable.
 Profiler (continuous): CPU/heap muestreado en producción sin overhead — flame graphs reales de hotspots.
 Ambos casi gratis de instrumentar con Cloud Ops agents/client libs (Java: auto-instrumentación Micrometer Tracing).

INTERMEDIO

GCP

Observabilidad y SRE

G171

Alertas que no despiertan a nadie en vano: higiene

INTERMEDIO

GCP

Observabilidad y SRE

G172

Golden signals: qué monitorear en TODO servicio

INTERMEDIO

GCP

Observabilidad y SRE

G173

SLOs en Cloud Monitoring: del concepto al setup

AVANZADO

GCP

Observabilidad y SRE

G174

Managed Service for Prometheus y Grafana en GCP

AVANZADO

GCP

Observabilidad y SRE

G175

Log analytics y queries sobre logs en BigQuery

AVANZADO

GCP

gcloud y herramientas

G176

Console: atajos y funcionalidades que la gente no conoce

BASICO

GCP

gcloud y herramientas

G177

Comandos gcloud del día a día (chuleta)

INTERMEDIO

GCP

gcloud y herramientas

G178

gsutil vs gcloud storage: la sucesión

INTERMEDIO

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

GCP ★
RESPUESTA

Observabilidad y SRE

G172

Latency (p50/p95/p99 por endpoint), Traffic (req/s), Errors (4xx/5xx rate), Saturation (CPU, memoria, conexiones pool, profundidad de colas).
 Por servicio: 4 dashboards que responden 'está sano?'.
 Dependencias: latencia y tasa de error de llamadas externas (son tus cuellos más frecuentes).

INTERMEDIO

GCP
RESPUESTA

Observabilidad y SRE

G171

Alertar sobre SÍNTOMAS que afectan usuarios (SLO burn, error rate), no causas triviales (CPU alta transitoria).
 Ventanas de estabilización (for 5 min), thresholds basados en p95, escalamiento jerárquico, runbook link en cada alerta.
 Revisar alertas disparadas al mes: <50% accionables = refinar.
 El objetivo: cada página = acción clara o se elimina.

INTERMEDIO

GCP
RESPUESTA

Observabilidad y SRE

G174

Managed Prometheus: ingesta de métricas de Prometeo gestionada (pagas por samples), sin operar servers — GKE lo scrapea automático.
 Cloud Monitoring soporta PromQL.
 Grafana: hosted por Google en Cloud Monitoring (dashboards de PromQL) o self-hosted en GKE contra el datasource managed.
 Migración típica: Prometheus self-hosted → managed para quitar ops.

AVANZADO

GCP
RESPUESTA

Observabilidad y SRE

G173

SLI: métrica de servicio percibida (disponibilidad = % requests 5xx-free; latencia = % requests < 300ms).
 SLO: objetivo en ventana (99.9% / 30 días rolling). Error budget = 100% - SLO.
 Monitoring: crear SLO sobre service → burn rate alerts (rápido 14.4x/1h y lento 6x/6h).
 Aterrizaje: pausa de releases cuando el budget está agotado (política de equipo).

AVANZADO

GCP
RESPUESTA

gcloud y herramientas

G176

Search global (barra superior) encuentra recursos por nombre/ID.
 Active Assist: recommendations centralizadas.
 Network Analyzer / Connectivity Tests.
 Cloud Console mobile app para alertas/uptime on-call.
 Keyboard shortcuts (?) y filtros guardados en tablas.
 Recent projects: cambiar rápido entre entornos.

BÁSICO

GCP
RESPUESTA

Observabilidad y SRE

G175

Log Analytics (bucket de logs con Linked dataset): consultar logs en BigQuery con SQL — tendencias de errores, latencia por endpoint desde LOGS (no métricas).
 Ejemplo: `SELECT httpRequest.requestUrl, COUNT(*) FROM logs WHERE severity='ERROR' GROUP BY 1 ORDER BY 2 DESC`.
 Retención extendida barata vs default 30 días.
 Base de análisis de incidentes post-mortem.

AVANZADO

GCP
RESPUESTA

gcloud y herramientas

G178

gsutil (legado) → gcloud storage (reemplazo moderno): `cp -r, rsync (delta), ls -l gs://bucket/prefijo**, mv, rm -a` (todas las versiones), `sign-url generate`.
 Patrones con wildcards y `--recursive`; `--billing-project` para requests en nombre de otro proyecto.
 Para pipelines: client libs o Storage Transfer, no shell.

INTERMEDIO

GCP
RESPUESTA

gcloud y herramientas

G177

`gcloud config set project/set compute/region`.
`run: gcloud run deploy svc --image img --region us-central1; run services update-traffic`.
`compute: instances create/list, ssh NAME --tunnel-through-iap`.
`sql: connect INSTANCE --user=root; sql backups create`.
`iam: auth list, auth login --update-adc; projects get-iam-policy`.
`--format/--filter/--quiet` para scripts.

INTERMEDIO

GCP

gcloud y herramientas

G179

Logs y debugging desde CLI

INTERMEDIO

GCP

gcloud y herramientas

G180

IAM desde CLI: inspección y simulación

INTERMEDIO

GCP

gcloud y herramientas

G181

Cloud Code / Cloud Workstations: desarrollo en el cloud

INTERMEDIO

GCP

gcloud y herramientas

G182

Costos por CLI y presupuesto programático

INTERMEDIO

GCP

gcloud y herramientas

G183

Cloud Asset Inventory: saber qué tienes corriendo

INTERMEDIO

GCP

Patrones de arquitectura

G184

Arquitectura canónica: app web moderna en GCP

INTERMEDIO

GCP

Patrones de arquitectura

G185

Backend for a mobile app: qué cambia en el diseño

INTERMEDIO

GCP

Patrones de arquitectura

G186

Multi-tenant SaaS en GCP: 3 modelos de aislamiento

INTERMEDIO

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

GCP

gcloud y herramientas

G180

RESPUESTA

gcloud projects get-iam-policy PROJECT --format=json.
get-ancestors-iam-policy (quién me da este permiso).
Policy Troubleshooter (console o API): '¿por qué no puedo X?' — evalúa el binding real (condiciones incluidas).
Recommender IAM: permisos no usados en 90 días → sugerencia de roles más ajustados.

INTERMEDIO

GCP

gcloud y herramientas

G179

RESPUESTA

gcloud logging read 'resource.type=cloud_run_revision AND severity>=ERROR' --limit 20 --format json.
gcloud run services logs read svc --region ... (tail -f).
gcloud compute instances get-serial-port-output (boot failures).
Streaming en console (Log Router live tail) para deploys en caliente.

INTERMEDIO

GCP

gcloud y herramientas

G182

RESPUESTA

gcloud billing accounts list; budgets create con --billing-account --budget-amount --threshold-rule.
BigQuery billing export queries para reportes custom por label.
API Cloud Billing Catalog para estimar precios en tooling.
Alerta 100% + acción: función que apaga entornos dev de noche (scheduler + script) — ahorro automático.

INTERMEDIO

GCP

gcloud y herramientas

G181

RESPUESTA

Cloud Code (extensión IDE): deploy/debug de Cloud Run y GKE desde VSCode/IntelliJ, scaffold integrado.
Cloud Workstations: entornos dev gestionados en la nube (CPU/RAM configurables, preinstalado, efímeros) — onboarding en minutos, código nunca en laptops.
Emuladores locales: pubsub-emulator, datastore, bigquery (bq beta emulators) para tests sin GCP real.

INTERMEDIO

GCP

Patrones de arquitectura

G184

RESPUESTA

Cloud DNS + Cloud LB (global HTTPS) → Cloud Run (API) + GCS/CDN (frontend) → Cloud SQL (HA) + Memorystore (caché) → Secret Manager/IAM.
Observabilidad: Cloud Ops. CI/CD: Cloud Build + Artifact Registry.
IaC: Terraform.
Escala: Run auto, SQL read replica si hace falta. DR: backup cross-region.
Es el punto de partida del 80% de proyectos.

INTERMEDIO

GCP

gcloud y herramientas

G183

RESPUESTA

Inventario de TODOS los recursos de la org en un punto del tiempo + histórico de cambios (feed a BigQuery/PubSub).
Queries tipo: 'todas las VMs con IP pública', 'buckets sin CMEK', 'SAs con keys'.
Base de compliance continua y de limpieza de recursos zombis.
gcloud asset search-all-resources --asset-types=...

INTERMEDIO

GCP

Patrones de arquitectura

G186

RESPUESTA

Pool (todos los tenants en la misma BD con tenant_id + row-level security) — barato.
Silos (BD/proyecto por tenant) — máximo aislamiento, caro de operar a escala.
Bridge (BD compartida, esquema o servicio por tenant grande).
Híbrido realista: pool para pequeños, silo para enterprise (cada silo es un proyecto con Terraform).

INTERMEDIO

GCP

Patrones de arquitectura

G185

RESPUESTA

Auth de usuarios finales: Identity Platform/Firebase Auth (JWT en headers).
APIs optimizadas para móvil: payloads pequeños, paginación por cursor, batching en BFF.
Push: FCM. Offline: Firestore local cache.
CDN para assets, Cloud Run para API — el tráfico móvil es spiky: serverless absorbe picos sin pagos fijos.

INTERMEDIO

GCP

Patrones de arquitectura

G187

API management: API Gateway vs Apigee vs Endpoints

INTERMEDIO

GCP

Patrones de arquitectura

G188

Exponer Cloud Run con dominio propio: opciones

INTERMEDIO

GCP

Patrones de arquitectura

G189

Webhooks entrantes (pagos, mensajería): diseño robusto

INTERMEDIO

GCP

Patrones de arquitectura

G190

Jobs nocturnos y batch sin servidores: opciones y criterio

INTERMEDIO

GCP

Escenarios de entrevista

G191

«Tu Cloud Run API se cae a las 3 a.m.» — runbook de diagnóstico

INTERMEDIO

GCP

Escenarios de entrevista

G192

«Migrar una app on-prem con BD a GCP» — camino realista

INTERMEDIO

GCP

Escenarios de entrevista

G193

«Otro equipo debe consumir nuestros datos» — cómo exponerlos bien

INTERMEDIO

GCP

Escenarios de entrevista

G194

«¿Compute Engine o Cloud Run para un microservicio Java?» — respuesta estructurada

INTERMEDIO

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

GCP

Patrones de arquitectura

G188

RESPUESTA

- 1) Domain mapping directo de Cloud Run (simple, sin LB).
 - 2) LB global + serverless NEG → Cloud Armor + CDN + certificado gestionado (el estándar para prod serio).
 - 3) Firebase Hosting rewrites a Run (frontend + API juntos).
- DNS: CNAME/A al endpoint. TTL bajo durante cutover.

INTERMEDIO

GCP

Patrones de arquitectura

G187

RESPUESTA

API Gateway (serverless, barato): auth JWT, routing a Cloud Run/Functions, quotas — para APIs simples.

Apigee (enterprise): portal de desarrolladores, monetización, analítica por producto de API, políticas avanzadas — para exponer APIs COMERCIALES.

Endpoints (legado, quizá evitar en nuevo).

Puerta de entrada + WAF (Cloud Armor) completa el borde.

INTERMEDIO

GCP

Patrones de arquitectura

G190

RESPUESTA

Cloud Run Jobs: contenedor batch, paralelismo por índice, retry por tarea — el default moderno.

Cloud Scheduler + Tasks: encadenar pasos con rate control.

Batch service: para cómputo HPC masivo (miles de VMs spot).

Dataflow: si es procesamiento de datos continuo/paralelo real.

Criterio: duración, paralelismo, dependencias.

INTERMEDIO

GCP

Patrones de arquitectura

G189

RESPUESTA

Endpoint idempotente por event id (dedup table) → aceptar 200 RÁPIDO y encolar el procesamiento (Pub/Sub/Tasks) → worker procesa con reintentos.

Verificación de firma del proveedor (HMAC) ANTES de aceptar.

Replay safety: proveedores re-envían; tu sistema debe tolerarlo.

Monitoreo de latencia y tasa de fallo por proveedor.

INTERMEDIO

GCP

Escenarios de entrevista

G192

RESPUESTA

Fase 0: inventario y assessment (compatibilidad, tamaño, latencia).

Lift & shift: Compute Engine + Cloud VPN/Interconnect + DMS para la BD (CDC, cutover controlado).

Después (no en paralelo): contenerizar → Cloud Run/GKE, reemplazar componentes por managed (Secrets, Storage).

Cutover por dominios con rollback planeado. Nunca modernizar y migrar a la vez.

INTERMEDIO

GCP

Escenarios de entrevista

G191

RESPUESTA

- 1) Dashboards: error rate, latencia, instancias, saturación.
- 2) Logs con traceId de requests fallidos → excepción dominante.
- 3) Dependencias: estado de Cloud SQL (failover?), límites de cuota (429?), circuit breakers abiertos.
- 4) Deploy reciente? → rollback de revisión (1 comando) mientras se investiga.
- 5) Postmortem sin culpa + alerta que debió dispararse antes.

INTERMEDIO

GCP

Escenarios de entrevista

G194

RESPUESTA

Preguntar primero: estado, tráfico (spiky?), equipo, licencias.

Default Cloud Run: contenedor, autoscaling, revisiones/canary gratis, sin parcheo de SO.

Compute Engine si: JVM tuning profundo (heap grande, GC específico), licencias por core/nodo, red especial (multicast), sidecars del sistema.

GKE si ya hay plataforma k8s y muchos servicios con mesh.

Justifica con costo operativo, no con preferencia.

INTERMEDIO

GCP

Escenarios de entrevista

G193

RESPUESTA

Opciones por acoplamiento: API REST/gRPC directa (sync, acopla), Pub/Sub topic con schema (eventos, desacopla), vista/materialized en BigQuery con data sharing (analítica), Data Share/Analytics Hub para datasets gobernados.

Contrato: schema versionado + SLA + auth por SA del consumidor con rol mínimo (publisher/invoker).

Nunca: acceso directo a TU base de datos (acoplamiento letal).

INTERMEDIO

GCP

Escenarios de entrevista

G195

«Auditoría: quién vio/exports datos de clientes» — implementación

INTERMEDIO

GCP

Patrones de arquitectura

G196

Procesamiento de archivos que suben usuarios: patrón completo

AVANZADO

GCP

Patrones de arquitectura

G197

Aplicación multi-región activa-activa: checklist

AVANZADO

GCP

Patrones de arquitectura

G198

Hybrid y multi-cloud: Anthos y el sentido real

AVANZADO

GCP

Patrones de arquitectura

G199

CQRS aplicado: lectura escalada con BigQuery/Firestore

AVANZADO

GCP

Patrones de arquitectura

G200

Anti-patrones GCP frecuentes (y cómo se ven en la factura)

AVANZADO

GCP

Escenarios de entrevista

G201

«Diseñe un sistema de notificaciones para 10M usuarios» — boceto GCP

AVANZADO

GCP

Escenarios de entrevista

G202

«Necesito garantizar que facturas NUNCA se pierden» — patrón

AVANZADO

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

GCP

Patrones de arquitectura

G196

RESPUESTA

1) Backend firma V4 signed URL → cliente sube directo a GCS (no pasa por la API).
 2) Notificación: Eventarc (GCS finalize) o Pub/Sub → Cloud Run procesa (valida, transformar, guarda metadata en SQL/Firestore).
 3) Estado y errores: tabla de jobs + Cloud Tasks retry.
 Ventajas: API ligera, escala automática, resumable uploads nativos.

AVANZADO

GCP

Escenarios de entrevista

G195

RESPUESTA

Data Access audit logs (ADMIN_READ/DATA_READ) habilitados para los servicios de datos sensibles → sink a BigQuery.
 Queries: por objeto accedido, por usuario, por método (get/download).
 Alertas: log-based metric para accesos masivos anómalos (exfil detection).
 Aplicación: tabla de acceso a PII con propósito (por qué lo vio) si el compliance lo exige.

INTERMEDIO

GCP

Patrones de arquitectura

G198

RESPUESTA

Anthos/GKE Enterprise: plano de gestión para GKE on-cloud, on-prem y otros clouds — políticas, config sync y service mesh unificados.
 Casos reales: migración gradual, cumplimiento de datos on-prem, evitar vendor lock-in de plataforma.
 Costo: licencias + complejidad operativa significativa — justificar con requisito concreto, no con moda.

AVANZADO

GCP

Patrones de arquitectura

G197

RESPUESTA

Datos: Spanner multi-región (o SQL cross-region replicas manual), GCS multi-region, Memorystore regional por región.
 Tráfico: LB global anycast → Cloud Run en 2+ regiones (health checks regionales).
 Estado de sesión: stateless + JWT; caché por región.
 DNS/routing: LB decide por latencia; failover automático.
 Pruebas: caos regional simulado cada trimestre.

AVANZADO

GCP

Patrones de arquitectura

G200

RESPUESTA

VMs sin apagar en dev (24/7 por hábito) → CUD de cosas que no necesitas.
 Logs ruidosos sin exclusiones → Cloud Logging caro.
 Egress directo de GCS a usuarios sin CDN → transfer cost alto.
 NAT de 1 instancia pero min-instances altas en Run.
 BigQuery sin particionar → scans de TB por dashboard.
 Cada uno tiene fix mecánico — auditar mensualmente.

AVANZADO

GCP

Patrones de arquitectura

G199

RESPUESTA

Escrituras en Cloud SQL (transaccional, normalizado) → CDC/Dataflow → vistas denormalizadas en Firestore (consultas de app) o BigQuery (analítica).
 El servicio de escritura no compite con lecturas masivas; cada lado optimiza su patrón.
 Lag eventual de segundos: aceptable para feeds/dashboards, no para saldo bancario.

AVANZADO

GCP

Escenarios de entrevista

G202

RESPUESTA

Escritura transaccional a Cloud SQL (factura + outbox event en la MISMA tx).
 Publisher outbox (poller/CDC Debezium) → Pub/Sub con retention + DLQ.
 Consumidor idempotente persiste en destino; reconciliación diaria compara conteos origen/destino.
 Backups PITR + retention de GCS = durabilidad de datos; el outbox resuelve la entrega.

AVANZADO

GCP

Escenarios de entrevista

G201

RESPUESTA

API recibe evento → Pub/Sub (topic notificaciones) → Dataflow/Cloud Run workers por canal: FCM (push), SendGrid (email), SMS — cada canal con rate limiting propio (Cloud Tasks).
 Preferencias de usuario en Firestore (por usuario, lectura rápida).
 Dedup + throttling por usuario.
 Métricas: entregados/fallidos por canal; DLQ para fallos; reintentos.
 Escala: fan-out por shards (region/hash) para no hotspotear.

AVANZADO

GCP

Escenarios de entrevista

G203

«Reduzca el 40% del costo sin tocar features» — plan de ataque

AVANZADO

GCP

Escenarios de entrevista

G204

«La app es lenta solo para usuarios de otro continente» — diagnóstico

AVANZADO

GCP

Escenarios de entrevista

G205

«Cumpló GDPR/PII en GCP» — controles concretos

AVANZADO

GCP

Escenarios de entrevista

G206

«Necesito un chat en tiempo real» — componentes GCP

AVANZADO

GCP

Escenarios de entrevista

G207

«Buscador sobre millones de documentos» — dónde entra cada servicio

AVANZADO

GCP

Escenarios de entrevista

G208

«Zero downtime deploy» — opciones por plataforma GCP

AVANZADO

GCP

Escenarios de entrevista

G209

«Rate limiting global para una API multi-región» — diseño

AVANZADO

GCP

Patrones de arquitectura

G210

Patrón claim check: payloads grandes por mensajería

AVANZADO

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

GCP

Escenarios de entrevista

G204

RESPUESTA

Medir: ¿latencia de red (TTFB) o de aplicación (server time)?
RUM/ping desde esa región.
Red: LB global anycast + CDN para estáticos (edge caching) — el fix más común.
Datos: BD en una sola región = RTT por query → read replica regional o caché local (Memorystore) de datos calientes.
Verificar handshake TLS y keep-alive (conexiones nuevas por request matan latencia).

AVANZADO

GCP

Escenarios de entrevista

G203

RESPUESTA

- 1) Billing export: top SKUs y por label/equipo — dónde está el dinero REAL.
- 2) Quick wins: dev apagado de noche (scheduler), IPs estáticas huérfanas, snapshots viejos, logs con exclusion.
- 3) Compute: right-sizing con Recommender, spot en batch, CUDs sobre la base estable.
- 4) Datos: storage classes por edad, particionado BQ, CDN para egress.
- 5) Medir y repetir; presupuesto + alertas para sostener.

AVANZADO

GCP

Escenarios de entrevista

G206

RESPUESTA

Transporte: WebSocket con STOMP (GKE) o SSE + polling (Cloud Run) — mantener conexiones requiere always-allocated CPU en Run o websockets en GKE.
Fan-out: Pub/Sub entre instancias/réplicas (todos los pods reciben el mensaje del room).
Persistencia: Firestore (realtime listeners para clientes web/móvil gratis).
Presencia y rate limiting: Memorystore.
Firebase Realtime/Firestore directo simplifica si el equipo es pequeño.

AVANZADO

GCP

Escenarios de entrevista

G205

RESPUESTA

Residencia: org policy resourceLocations + regiones de datos de cada servicio.
Acceso: IAM mínimo + VPC-SC (perímetro) + CMEK si aplica.
Ciclo de vida: retención automática (GCS lifecycle, BQ table expiration), derecho al olvido = jobs de borrado trazables.
Detección: Sensitive Data Protection escaneando buckets/tablas.
Auditoría: data access logs habilitados y exportados.

AVANZADO

GCP

Escenarios de entrevista

G208

RESPUESTA

Cloud Run: revisiones + traffic split (canary 5% → 100%), rollback instantáneo.
GKE: rolling update (maxSurge/maxUnavailable) o canary con Gateway API/Istio.
MIG: rolling updater con health checks.
Base de datos: migraciones compatibles (expand → deploy → contract) para que N y N+1 convivan.
La parte dura casi siempre es el SCHEMA, no el deploy.

AVANZADO

GCP

Escenarios de entrevista

G207

RESPUESTA

Ingesta: GCS + Eventarc → Dataflow (parse/enrich) → índice.
Búsqueda full-text gestionada: Elastic Cloud en Marketplace, Vertex AI Search (antes Enterprise Search), o pg_trgm/tsvector en Cloud SQL para escala chica.
Embeddings: Vertex AI text-embedding + vector search (Matching Engine) para semántica.
Queries: API en Cloud Run con caché de resultados calientes.

AVANZADO

GCP

Patrones de arquitectura

G210

RESPUESTA

Problema: mensajes >10MB (imágenes, PDFs) no van bien en Pub/Sub.
Solución: guardar payload en GCS → publicar mensaje solo con la REFERENCIA (bucket, objeto, metadata) → consumidor descarga y procesa.
Beneficios: mensajería liviana y rápida, retry barato, lifecycle del payload separado del mensaje.
Nombre clásico: Claim Check (Enterprise Integration Patterns).

AVANZADO

GCP

Escenarios de entrevista

G209

RESPUESTA

Límite por API key/usuario: contador distribuido en Memorystore (Redis) — pero Redis por región ≠ global consistente.
Opciones: Redis global vía replicas + read local (aproximado), contador en Spanner (consistente, costo), o rate limit por-región (suficiente en la práctica).
En el borde: Cloud Armor rate-based por IP (grueso) + fino por identidad en la app.
Documentar el trade-off de precisión.

AVANZADO

GCP

Patrones de arquitectura

G211

Compensación y reconciliación diaria: la red de seguridad

AVANZADO

GCP

Escenarios de entrevista

G212

«Justifica tu arquitectura ante un CFO» — lenguaje de costo

AVANZADO

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

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

GCP

RESPUESTA

Escenarios de entrevista

G212

Traduce a dinero: 'Cloud Run escala a 0 → pagamos uso real vs \$X/mes de VMs siempre encendidas; ahorro estimado Y%'.
TCA: infraestructura + operación (horas de DevOps ahorradas por managed) + riesgo (SLA/fallas evitadas).
Costo por unidad de negocio: \$ por pedido procesado/por usuario activo — la métrica que entiende todos.
CUDs y compromisos = descuento contra estabilidad de demanda.

AVANZADO

GCP

RESPUESTA

Patrones de arquitectura

G211

Aun con outbox/idempotencia, los sistemas divergen (bugs, mensajes perdidos).
Job de reconciliación: compara conteos/sumas entre sistemas (pedidos vs pagos del día), reporta diferencias, genera work-queue de reparación.
Métrica: % de records divergentes por día — tendiente a 0.
Es humilde pero es lo que mantiene la confianza en event-driven.

AVANZADO

MICROSERVICIOS

★

RESPUESTA

Fundamentos y decisiones

M002

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

★

RESPUESTA

Fundamentos y decisiones

M001

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

RESPUESTA

Fundamentos y decisiones

M004

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

★

RESPUESTA

Fundamentos y decisiones

M003

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

RESPUESTA

Fundamentos y decisiones

M006

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

RESPUESTA

Fundamentos y decisiones

M005

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

MICROSERVICIOSFundamentos y decisionesM007

Ley de Conway aplicada: tu arquitectura imita tu organización

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM008

Event Storming: qué es y qué produce

INTERMEDIO

MICROSERVICIOS★Fundamentos y decisionesM009

Strangler Fig: migrar el monolito sin big bang

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM010

Distributed monolith: los síntomas del fracaso

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM011

ADR: Architecture Decision Records

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM012

Service ownership: qué significa end-to-end

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM013

Métricas DORA: las 4 que miden entrega de software

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM014

¿Cuántos microservicios son demasiados?

INTERMEDIO

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

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

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

MICROSERVICIOSFundamentos y decisionesM015

C4 model: documentar arquitectura en 4 niveles

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM016

API-first: qué significa de verdad

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM017

Domain services vs application services

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM018

Anticorruption Layer: cuándo y cómo

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM019

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

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM020

Cuándo EXTRAER un servicio del monolito: señales concretas

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM021

Product over project: el cambio de mentalidad

INTERMEDIO

MICROSERVICIOSFundamentos y decisionesM022

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

INTERMEDIO

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

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

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 ($A \rightarrow B \rightarrow C \rightarrow D$ 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

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

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

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

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

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

MICROSERVICIOSFundamentos y decisionesM031

AVANZADO

Agregados DDD: la unidad de consistencia

MICROSERVICIOS★Fundamentos y decisionesM032

AVANZADO

Context Mapping: relaciones entre bounded contexts

MICROSERVICIOSFundamentos y decisionesM033

AVANZADO

Hexagonal/Ports & Adapters dentro de un microservicio

MICROSERVICIOSFundamentos y decisionesM034

AVANZADO

Arquitectura evolutiva: fitness functions

MICROSERVICIOSFundamentos y decisionesM035

AVANZADO

Branch by abstraction: refactor sin big bang

MICROSERVICIOS★ComunicaciónM036

INTERMEDIO

Comunicación síncrona vs asíncrona

MICROSERVICIOS★ComunicaciónM037

INTERMEDIO

REST vs gRPC

MICROSERVICIOS★ComunicaciónM038

INTERMEDIO

Service Discovery: ¿por qué y cómo?

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

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

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

Quando 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

M039

Event notification vs event-carried state transfer

INTERMEDIO

MICROSERVICIOS

Comunicación

M040

Request-reply sobre mensajería: cómo y cuándo

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

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

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

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 + deprecate), 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 reintentará a B 3 veces; B reintentará 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

M047

Correlation ID vs message ID vs causation ID

INTERMEDIO

MICROSERVICIOS

Comunicación

M048

CloudEvents: el sobre estándar de eventos

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

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

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

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

M055

Message Router / Content-based routing: qué es

INTERMEDIO

MICROSERVICIOS

Comunicación

M056

Backpressure en sistemas de mensajería: quién frena

AVANZADO

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

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

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

M058

RESPUESTA

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

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.

BASICO

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

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

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

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

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

M071

Patrón Saga (transacciones distribuidas)

AVANZADO

MICROSERVICIOS★

Datos y consistencia

M072

¿Qué es CQRS?

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

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

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

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

M079

Saga coreografiada: ejemplo
pedido→pago→inventario

AVANZADO

MICROSERVICIOS

Datos y consistencia

M080

Saga orquestada: cuándo el coordinador
central gana

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

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

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

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

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

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

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

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

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

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

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

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

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

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

M103

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

INTERMEDIO

MICROSERVICIOS

Resiliencia

M104

Circuit breaker + caché: el combo de lectura resiliente

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

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

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

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

M111

Graceful degradation: la matriz de degradación

AVANZADO

MICROSERVICIOS

Resiliencia

M112

Chaos engineering: el experimento bien planteado

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

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

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

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 /actuator/metrics (resilience4j.circuitbreaker.state).

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

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

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

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

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 appender (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

M127

Dependencias externas: cómo observar lo que no controlas

INTERMEDIO

MICROSERVICIOS

Observabilidad

M128

Postmortem blameless: estructura que funciona

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

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

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

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

M135

¿Qué es un API Gateway y qué responsabilidades tiene?

INTERMEDIO

MICROSERVICIOS

★

APIs y edge

M136

Versionado de APIs sin romper clientes

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

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

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

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 (dataloaders obligatorios), 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

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

MICROSERVICIOS

Seguridad

M145

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

INTERMEDIO

MICROSERVICIOS

Seguridad

M146

Secrets en un ecosistema de microservicios: patrón completo

INTERMEDIO

MICROSERVICIOS

Seguridad

M147

PII entre microservicios: minimización y marcado

INTERMEDIO

MICROSERVICIOS

Seguridad

M148

Rate limiting por identidad vs por IP: cuándo cada uno

INTERMEDIO

MICROSERVICIOS

Seguridad

M149

Zero trust en una frase aplicado a microservicios

INTERMEDIO

MICROSERVICIOS

Seguridad

M150

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

INTERMEDIO

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

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

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

M151

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

AVANZADO

MICROSERVICIOS

Seguridad

M152

Autorización centralizada vs por servicio (OPA/sidecar)

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

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

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

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

M159

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

BASICO

MICROSERVICIOS

★

Plataforma y despliegue

M160

Contenedor vs máquina virtual: diferencia clave

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

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

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

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 (reiniciará 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 vigilas 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

M167

Feature flags: liberar ≠ desplegar

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M168

Ambientes efímeros (preview environments)

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

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

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

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

M175

Versionado de imágenes y trazabilidad de deploy

INTERMEDIO

MICROSERVICIOS

Plataforma y despliegue

M176

Consumidores de colas: despliegue y compatibilidad

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

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

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

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

M183

Migraciones de schema compatibles: el contrato entre deploys

AVANZADO

MICROSERVICIOS

Plataforma y despliegue

M184

Canary automated: qué métricas deciden promover

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

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

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

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

M191

Contract testing consumer-driven (Pact): el flujo

AVANZADO

MICROSERVICIOS

Testing y calidad

M192

Testing en producción: formas seguras

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

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

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

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 zombies pagando infra y parches = deuda silenciosa.

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

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

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

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