

UNIVERSIDAD NACIONAL DE INGENIERÍA

Arquitectura de Software

Arquitecturas reales y cierre del curso

Cómo se lee un caso · Stack Overflow, Discord y Figma · El curso en una página · Presentar y defender

Gerson Garrido

Semana 16

Agenda

Sesión A

1. Cómo se lee un caso
2. Stack Overflow, Discord y Figma
3. Los tres, lado a lado
4. El curso en una página
5. El rol

Sesión B y laboratorio

6. Cómo se presenta una arquitectura
7. Criticar y defender con escenarios
8. Una falla elegida en el momento
9. La defensa, paso a paso

Lectura

Una, a elección: Craver, *Stack Overflow: The Architecture — 2016 Edition*; Ingram, *How Discord Stores Trillions of Messages* (2023); Steele, *How Figma's databases team lived to tell the scale* (2024). Complementarias: AOSA vol. 2, *nginx*; Westeinde, *Deconstructing the Monolith* (2019); *Software Architecture in Practice*, cap. 24 y 25; Fowler, *Who Needs an Architect?*

SECCIÓN 01

Cómo se lee un caso

01

Esta semana no hay técnica nueva

Leer

La arquitectura de un sistema ajeno, contada por quienes lo operaban.

Tres textos, con números y con problemas.

Explicar

La arquitectura de un sistema propio: su Tinkuy.

Qué se decidió, qué se descartó y cómo se comprobó.

Las dos son el trabajo

Fuera de un curso, buena parte de hacer arquitectura es leer sistemas de otros y explicar el propio.

Un caso no es un libro de texto

- Lo escribe alguien que trabajó en el sistema.
- Cuenta lo que pasó, en el orden que le conviene a quien lo cuenta.
- Tiene un propósito que rara vez es enseñar: mostrar un buen trabajo, atraer gente al equipo.

No es falso: es incompleto de forma predecible

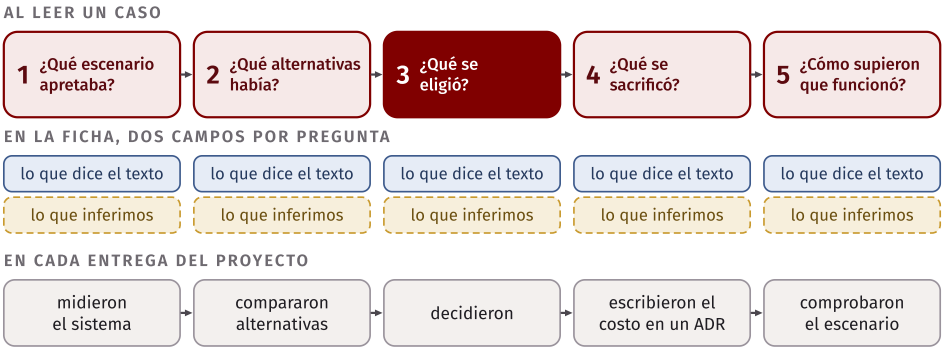
Cuenta más la solución que el costo. Por eso se lee con preguntas preparadas.

Las cinco preguntas

1. ¿Cuál era el **escenario que apretaba**? *(carga, crecimiento, fallas, velocidad de cambio)*
2. ¿Qué **alternativas** había? Incluida la de no hacer nada.
3. ¿Qué se **eligió**?
4. ¿Qué se **sacrificó**?
5. ¿Cómo **supieron** que funcionó?

> **Es el ciclo del proyecto, visto desde afuera**
>
> Medir, comparar, decidir, escribir el costo, comprobar. Un texto que solo responde la tercera no es un caso: es publicidad.

Las cinco preguntas, en la ficha y en el proyecto



Un texto que solo responde la 3, sin alternativas, sin costos y sin medición, es publicidad.

La 4 es la que los textos responden peor, y la que más enseña.

Lo que el texto dice y lo que uno infiere

Lo que el texto afirma

«El p99 quedó en 15 ms».
«Tardamos nueve meses en la primera tabla».

Se cita, con su fuente y su fecha.

Lo que uno infiere

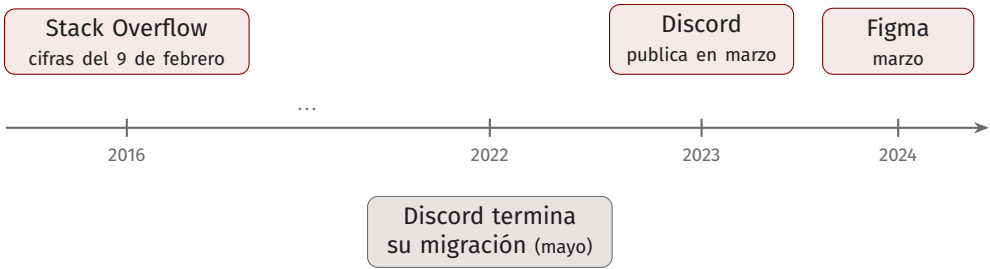
«Esto es un bulkhead».
«Por eso no usaron microservicios».

Es de quien lee, y se dice así.

Un ejemplo de cómo se resbala

El texto de Stack Overflow no usa la palabra «monolito» en ningún lugar. «Es un monolito» es una inferencia razonable. «El texto dice que es un monolito» es falso.

Un caso es una foto con fecha



- Cada texto describe su sistema en el momento en que se escribió.
- Ninguno dice cómo es hoy. Estas diapositivas tampoco.
- Lo que se aprende de una foto es el razonamiento, no la tecnología.

La ficha de lectura

case-worksheet-template.md

```
## 4. Lo que se sacrificó

- **Lo que dice el texto:** <qué costo reconoce>
- **Lo que inferimos:** <costos que el texto no menciona>
```

- Los datos del texto, **con su fecha**; las cinco preguntas, con dos campos cada una.
- Al final: lo que el texto no dice, las semanas del curso que ilustra y qué no copiarían sin su contexto.

make check

```
node check-doc.ts worksheet example/case-worksheet-shopify.md
✓ example/case-worksheet-shopify.md
```

Una ficha que solo cuenta la solución

```
make errors
```

```
✗ errors/worksheet-only-solution.md
    El texto: falta el campo "Qué describe"
    El texto: la fecha no dice el año: un caso describe su sistema en un momento
    1. El escenario que apretaba: falta el campo "Lo que dice el texto"
    1. El escenario que apretaba: falta el campo "Lo que inferimos"
    2. Las alternativas: falta la sección
    4. Lo que se sacrificó: falta la sección
    Lo que el texto no dice: falta la sección
    Semanas del curso: no nombra ninguna semana ("semana 11")
    Qué no copiaríamos sin su contexto: falta la sección
check-doc terminó con código 1
```

- Faltan justo las dos preguntas que separan un caso de un anuncio.
- El revisor mira la forma. Si el texto se leyó bien lo revisa otro equipo.

SECCIÓN 02

Stack Overflow (2016)

02

Stack Overflow: lo que dice el texto

Un día: 9 de febrero de 2016

- 209 420 973 pedidos HTTP al load balancer
- 504 816 843 consultas SQL
- 5 831 683 114 accesos a Redis
- 22,71 ms en promedio para generar la página de una pregunta

Las máquinas

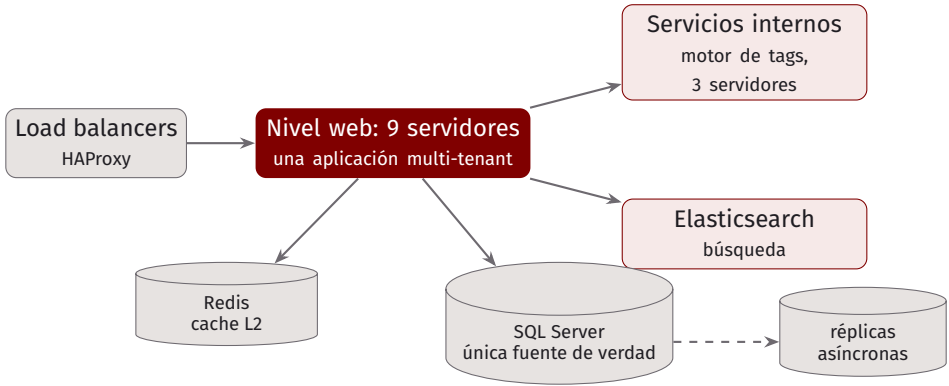
- 4 servidores de SQL Server
- 11 servidores web
- 2 de Redis
- 3 para el motor de tags
- 3 de Elasticsearch
- 4 load balancers

Es un inventario, no la historia de una decisión

El autor lo presenta como un panorama; varias de las cinco preguntas quedan a medio responder.

Nick Craver, *Stack Overflow: The Architecture — 2016 Edition*, 17 de febrero de 2016.

Stack Overflow: las piezas



- Todo es redundante, y hay un segundo centro de datos.
- El texto dice que el sitio podía funcionar con un solo servidor web, y que no por eso es buena idea.

Stack Overflow: cache en dos niveles

1. **L1**: la memoria de cada servidor web.
2. **L2**: Redis, común a todos.
3. Si no está en ninguna: se trae del origen y se guarda en las dos.
4. Cuando un servidor borra un valor, publica un mensaje y los demás lo quitan de su L1.

Semana 11

Es la respuesta de este sistema a los datos viejos en una cache local. La cache no es gratis: necesita invalidación.

Stack Overflow: las cinco preguntas

	Lo que dice el texto	Lo que inferimos
Escenario	Describe un sistema que funciona; da la carga de un día	De rendimiento. Da promedios, no percentiles
Alternativas	Solo puntuales: Elasticsearch o SQL; websockets o polling	No discute repartir en servicios ni en shards
Elección	Una aplicación, pocas máquinas, cache, redundancia	Un monolito con servicios alrededor; escala hacia arriba
Costo	Casi nada explícito	Hardware propio; un principal por clúster; afinación que depende de pocas personas
Evidencia	Más pedidos que en 2013 y menos tiempo de proceso: de 757 a 447 horas	Mezcla dos cambios: hardware y código

Stack Overflow: lo que el caso sostiene

Más de lo que dice el texto

«Stack Overflow demuestra que un monolito escala».

Lo que el texto muestra

En febrero de 2016, esa carga se atendía con esas máquinas. **El estilo no determina la escala.**

- Quien proponga distribuir un sistema mucho más chico «para que escale» tiene que decir qué escenario no cumple el diseño simple.
- Tener la forma de un sistema que escala no da su rendimiento: además de la forma, tenían años de medir cada pedido.

SECCIÓN 03

Discord (2023)

03

Discord: el problema

- 2017: 12 nodos de Cassandra.
- Comienzos de 2022: 177 nodos, billones de mensajes.
- Guardias frecuentes, latencia impredecible, mantenimiento cada vez más caro.

La clave de partición

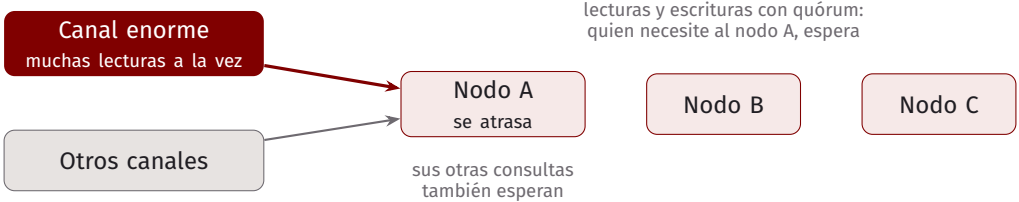
El canal y una ventana fija de tiempo. Todos los mensajes de un canal en una ventana quedan juntos, en tres nodos.

La consecuencia

Un servidor con cientos de miles de personas envía órdenes de magnitud más mensajes que uno de amigos: hay particiones muchísimo más pedidas que otras.

Bo Ingram, *How Discord Stores Trillions of Messages*, 6 de marzo de 2023.

Discord: una partición caliente degrada a todos



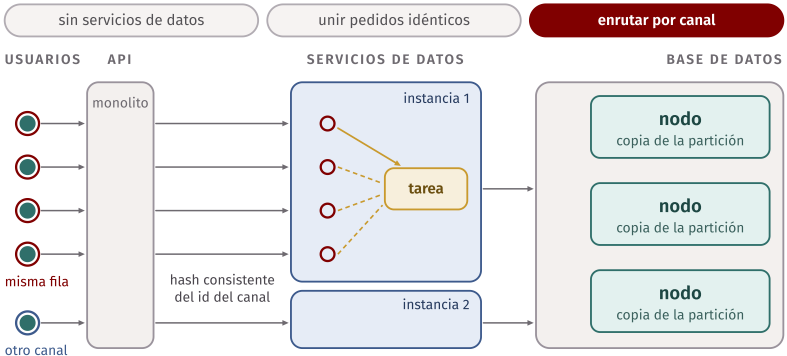
Semana 11

Un hot spot no se arregla con la función de hash: el problema es *una* clave. La clave reparte bien los canales, y aun así un canal enorme es una partición enorme de pedidos.

Discord: servicios de datos

- Entre la API y las bases de datos. Sin lógica de negocio.
- Si varios piden la misma fila a la vez, se consulta una vez.
- Delante, enrutamiento por hash consistente del canal: todos los pedidos de un canal van a la misma instancia.
- El texto es franco: ayudó, y las particiones calientes siguieron, menos seguido.
- **Inferimos**: es el remedio de la estampida (semana 11), delante de la base de datos.

Discord (2023): unir pedidos y enrutar por canal



Lo que cuenta el texto de Discord (2023). Las cantidades de usuarios y de instancias del dibujo son solo para ilustrar.

Discord: la migración

El plan

Gradual: lo nuevo a ScyllaDB desde una fecha de corte, lo histórico por detrás. Estimación del migrador existente: tres meses.

Lo que hicieron

Un migrador propio: nueve días. Abandonaron el plan gradual y cambiaron todo de una vez.

Lo que lo hizo aceptable, según el texto:

- **Doble escritura**: los mensajes nuevos iban a las dos bases de datos.
- **Copia con puntos de control**: se podía retomar.
- **Validación**: una fracción de las lecturas a las dos, comparando resultados.

El texto no dice si mantuvieron Cassandra como vuelta atrás después del cambio.

Discord: cómo supieron que funcionó

	Cassandra	ScyllaDB
Nodos	177	72
Disco por nodo	4 TB en promedio	9 TB
p99 de traer mensajes históricos	entre 40 y 125 ms	15 ms
p99 de insertar un mensaje	entre 5 y 70 ms	5 ms

- Reporta p99, no promedios (semana 2), y un pico real: la final del mundial de 2022.
- Inferimos: la comparación de nodos no es a igual capacidad, y no separa cuánto aportó el motor y cuánto los servicios de datos.

Cifras del texto de Discord (2023); el cambio fue en mayo de 2022.

Discord: lo que el texto dice y lo que no

Dice

El objetivo: controlar el tráfico concurrente hacia las particiones calientes.

Los mecanismos: unir pedidos idénticos y enrutar por canal.

No dice

No describe un límite de llamadas concurrentes ni un pool por canal: no describe un bulkhead (semana 14).

- Lo que sí muestra es el problema que un bulkhead ataca: un recurso compartido que, saturado por un cliente, degrada a los demás.
- **Trade-off** de unir pedidos: menos carga cuando más duele; a cambio, muchos pedidos dependen de una sola consulta.

SECCIÓN 04

Figma (2024)

04

Figma: el punto de partida y el límite

- 2020: una sola base de datos PostgreSQL, en la instancia más grande de su proveedor.
- Fines de 2022: cache, réplicas de lectura y una docena de bases de datos por **partición vertical** (grupos de tablas relacionadas, cada uno a su base de datos).
- El piso de la partición vertical: lo más pequeño que se puede mover es una tabla.
- Había tablas de varios terabytes y miles de millones de filas; el límite de escrituras estaba a meses.

El plazo es parte del escenario

Con dos años de margen, la alternativa de cambiar de motor se habría evaluado distinto.

Sammy Steele, *How Figma's databases team lived to tell the scale*, 14 de marzo de 2024.

Figma: criterios antes que opciones

Lo que una solución debía cumplir

- Afectar poco a quienes desarrollan el producto.
- No exigir copias masivas de datos.
- Poder avanzar de a pocos.
- Evitar migraciones de un solo sentido.
- Consistencia fuerte: sin doble escritura.

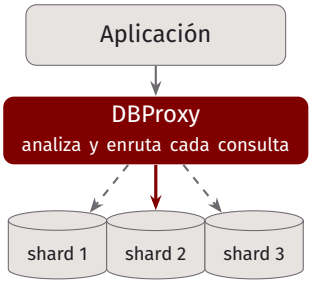
Las alternativas, y por qué no

- Otro motor con sharding incorporado: migración compleja y perder lo que sabían operar.
- NoSQL: reescribir casi todo el backend.
- **Elegida**: sharding propio sobre PostgreSQL, a la medida.

Semana 7

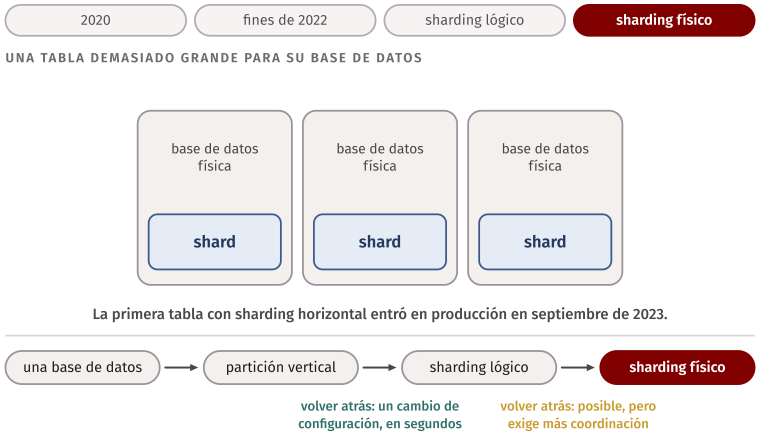
De los tres, es el texto que más se parece a un ADR.

Figma: lo que construyeron



- **Varias claves**, no una: usuario, archivo, organización. Las tablas que comparten clave se agrupan.
- **Hash de la clave**: los identificadores secuenciales habrían concentrado los datos. Costo: las consultas por rango.
- Una consulta sin la clave va a todos los shards.
- Qué consultas soportar: lo decidieron probando «en sombra» con tráfico real.

Figma (2024): el camino que cuenta el texto



Según el texto de Figma (2024). El número de shards del dibujo es solo para ilustrar.

Figma: primero el paso que se puede deshacer

- Con el sharding lógico, lecturas y escrituras se comportan como si la tabla ya estuviera repartida.
- Se prueba todo el camino antes del paso riesgoso (semanas 9 y 15).

Figma: lo que costó, dicho por el texto

- Sin transacciones atómicas entre shards: una escritura puede quedar a medias.
- `JOIN` solo dentro de un grupo y por la clave.
- Sin claves foráneas ni índices únicos globales.
- Cambios de esquema coordinados en todos los shards.
- Nueve meses para la primera tabla.

El resultado

Primera tabla en septiembre de 2023. Diez segundos de disponibilidad parcial en los principales; ninguna regresión.

Una frase poco común

Cuando tengan margen, van a reevaluar si siguen con su solución propia.

SECCIÓN 05

Los tres, lado a lado

05

Los tres casos, lado a lado

	Stack Overflow (2016)	Discord (2023)	Figma (2024)
Tipo de texto	Inventario del sistema	Un problema y su solución	Una decisión con criterios
Lo que apretaba	Nada urgente	Latencia y operación	Tablas que no cabían, con plazo
La respuesta	Pocas máquinas grandes, cache	Servicios de datos y otro motor	Sharding propio sobre lo que tenían
Costo reconocido	Casi nada	Poco	Mucho, con detalle
Evidencia	Cifras de un día	p99 y nodos	Pruebas previas y el primer cambio

Lo común y lo distinto

Lo que los tres sostienen

- Miden.
- Prefieren lo que su equipo sabe operar.
- Ninguno presenta su arquitectura como *la* correcta.

Donde difieren

- Stack Overflow no cuenta ninguna migración.
- Discord migró de una vez; Figma, de a pocos y con vuelta atrás.
- Discord usó doble escritura; Figma la descartó.

La misma técnica, buena en un contexto y descartada en otro

Un mensaje se escribe una vez y casi no cambia. Un modelo relacional con transacciones no tolera una escritura que entra en una base de datos y no en la otra.

Error común: copiar la arquitectura sin su contexto

- La de **Discord** responde a billones de mensajes y a 177 nodos que ya no podían mantener.
- La de **Stack Overflow**, a una carga dominada por lecturas y a un equipo que conocía cada detalle de sus máquinas.
- La de **Figma**, a tablas de terabytes y a un plazo de meses.

Qué se copia

La forma de decidir, no la decisión.

SECCIÓN 06

El curso en una página

06

El recorrido

Sem.	La pregunta	Sem.	La pregunta
1	¿Qué es la arquitectura y qué la determina?	10	¿Cómo se sobrevive a la caída de una máquina con datos?
2	¿Cómo se dice qué tan bien debe funcionar?	11	¿Y cuando no cabe en una máquina, o se lee demasiado?
3	¿Cuánto cuesta un cambio?	12	¿Cómo se mantiene correcto lo que ocurre a la vez?
4	¿Cómo se organiza un solo despliegue?	13	¿Cómo se comunican las partes sin esperarse?
5	¿Cuándo se distribuye?	14	¿Qué pasa cuando algo falla?
6	¿Con qué se mejora un atributo?	15	¿Cómo se mantiene todo esto en el tiempo?
7	¿Cómo se registra y se evalúa una decisión?	16	¿Cómo se lee una arquitectura ajena y se defiende la propia?
9	¿Qué cambia cuando los datos son el problema?		

Una sola pregunta

Sin contexto no hay respuestas correctas. Entonces, ¿cómo se decide?

Seis ideas

1. La arquitectura la determinan los **atributos de calidad**. (1 y 2)
2. La **modularidad** decide cuánto cuesta cambiar. (3 y 4)
3. **Distribuir** se paga con fallas parciales y con consistencia. (5, 9 a 13)
4. Cada **táctica** mejora un atributo y cuesta otro. (6)
5. Una **decisión** vale lo que valen su justificación y su comprobación. (7 y 15)
6. El sistema tiene que seguir funcionando **mientras algo falla**. (14)

Caben en una

Decidir con evidencia y dejar escrito lo que se pagó. Las tecnologías del curso van a cambiar; ese ciclo, no.

Cada semana, bajo la idea a la que sirve

DIECISÉIS SEMANAS



SEIS IDEAS

1	**La arquitectura la determinan los atributos de calidad**	semanas 1 y 2
2	**La modularidad decide cuánto cuesta cambiar**	semanas 3 y 4
3	**Distribuir se paga con fallas parciales y con consistencia**	semanas 5 y 9 a 13
4	**Cada táctica mejora un atributo y cuesta otro**	semana 6
5	**Una decisión vale lo que valen su justificación y su comprobación**	semanas 7 y 15
6	**El sistema tiene que seguir funcionando mientras algo falla**	semana 14

Semana 8: repaso de la primera mitad. Semana 16: leer una arquitectura ajena y defender la propia.

Lo que el curso no cubrió

- **Despliegue y operación** en detalle: pipelines, infraestructura como código, orquestación de contenedores.
- **La nube y sus costos**: en el curso ninguna decisión tuvo precio en dinero.
- **Los algoritmos** de los sistemas distribuidos: consenso, relojes.
- **La interfaz de usuario**: Tinkuy fue una API.
- **La seguridad** más allá de las tácticas.

> **Pregunta**
> ¿Qué le falta a su Tinkuy para atender a compradores de verdad?

SECCIÓN 07

El rol de quien hace arquitectura

07

Una actividad antes que un cargo

- Nadie en su equipo tuvo el cargo, y tomaron decisiones de arquitectura cada dos o tres semanas.
- Arquitectura es lo que el equipo **considera importante**, y las decisiones que quisiera acertar temprano porque **cuesta cambiarlas**.

Lo que la actividad exige:

- Entender el negocio lo suficiente para saber qué atributo importa.
- Conocer muchas opciones más que una a fondo.
- Comunicar: la mayor parte del trabajo es explicar y negociar.
- Seguir cerca del código, para que las decisiones no se separen de la realidad.

Martin Fowler, *Who Needs an Architect?*, IEEE Software, 2003, citando a Ralph Johnson.

Reducir lo irreversible

Fowler, 2003

Si arquitectura son las decisiones difíciles de cambiar, una de las tareas más importantes de quien la hace es **quitar arquitectura**: lograr que una decisión deje de ser irreversible.

- El medio de pago como plug-in (P1).
- Expandir, migrar y contraer (semana 9).
- El sharding lógico de Figma, antes del físico.

Trade-off

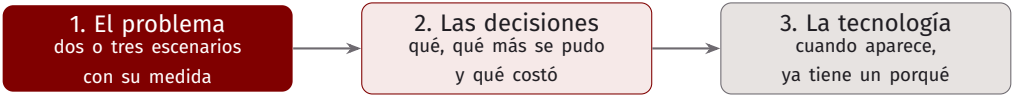
Cada cosa que se vuelve fácil de cambiar agrega complejidad. No se puede con todas a la vez: se paga donde el cambio es probable y equivocarse es caro.

SECCIÓN 08

Cómo se presenta una arquitectura

08

El orden: problema, decisiones, tecnología



Dos formas de empezar mal

Por la tecnología («usamos PostgreSQL, Redis y NATS»): nadie puede evaluar una respuesta sin la pregunta.

En orden cronológico («en P0 hicimos…»): es el orden en que llegaron los requisitos, no el que explica el sistema.

Un diagrama por idea

- Cada diagrama dice qué tipo de vista es: lo que corre y lo que se importa no se mezclan.
- Toda caja dice qué es y con qué tecnología; toda flecha, qué lleva y por qué protocolo.
- Con leyenda.
- El antes y el después, con la misma disposición: lo nuevo se ve sin leer.
- Para explicar una decisión bastan las tres o cuatro cajas que tocó.

Antes de presentar

Vuelvan a pasar `check-c4.ts` (semana 1) y comparen el diagrama con el sistema corriendo. Un diagrama que no coincide es lo primero que una pregunta descubre.

Qué se muestra de una decisión

presentation-template.md

```
### Decisión 1: <título, como el de su ADR>

- **Escenario:** <el escenario que la motivó, con su medida>
- **Alternativas descartadas:** <cuáles y por qué: una frase por alternativa>
- **Decisión:** <qué se hizo, en una frase, y en qué ADR está>
- **Lo que costó:** <qué atributo empeoró o qué se volvió más difícil>
- **Medición:** <antes y después, con el comando y la máquina>
```

- Las alternativas son lo que más se omite y lo que muestra que hubo una decisión.
- La medición, tal como salió: la misma máquina, antes y después. Se afirma la comparación.
- **Elegir tres** es parte del trabajo: ¿cuáles costaría más deshacer hoy?

Lo que haríamos distinto, la demostración y la deuda

Distinto

Qué sabían al decidir, qué saben ahora, qué harían.

«Elegimos mal» no dice nada.

Demostración

Decir antes qué se va a ver. Señalar la evidencia. Tener la salida de una corrida anterior.

Deuda

Escrita: qué es, qué cuesta hoy y qué la volvería urgente.

Una demostración sin predicción es una anécdota

Si no dijeron qué esperaban, cualquier resultado parece bueno.

SECCIÓN 09

Criticar y defender con escenarios

09

Criticar con escenarios, no con preferencias

Una preferencia

«Yo habría usado otra cosa».
«¿Por qué no microservicios?»

No se puede responder: no dice qué saldría mal.

Un escenario

«¿Qué pasa con este diseño si se abre la venta de cien conciertos a la misma hora?»

Se responde recorriendo el diseño, y la respuesta se puede comprobar.

Cuando les toque preguntar: conviertan la preferencia en escenario antes de hablar. Si no encuentran ninguno, era solo una preferencia.

Responder un «¿qué pasa si…?»

1. **Repetir la pregunta como escenario.** Fija el entorno; si falta un dato, se pregunta.
2. **Recorrerlo sobre el diagrama.** No de memoria. Ahí aparecen los recursos compartidos.
3. **Decir qué aguanta, qué no y por qué.** Las tres partes.
4. **Decir qué decisión habría que revisar.** Con su ADR.

> **Tres cosas más**
>
> «No lo sé» vale si viene con cómo lo averiguarían. Lo medido y lo esperado se dicen distinto. Si la pregunta encontró un hueco, se dice: es deuda.

Las familias de preguntas

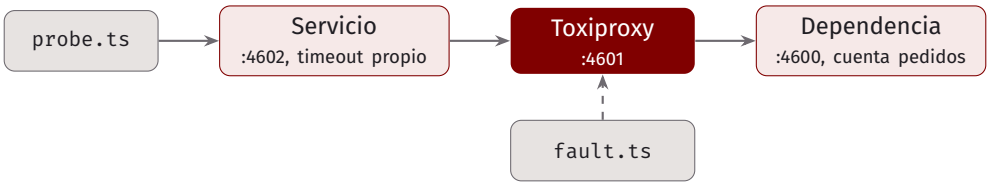
Familia	Ejemplo	Qué se mira
Más carga	¿Y cien conciertos a la vez?	Los recursos compartidos: qué se satura primero
Una dependencia falla	¿Y si la pasarela tarda diez segundos?	Quién la espera, con qué timeout
Una falla a la mitad	¿Y si el proceso muere después de cobrar?	El orden de las escrituras; qué es atómico
Datos viejos	¿Se ve libre un asiento ya vendido?	De dónde lee cada pantalla
Un requisito nuevo	Un segundo país con otra pasarela	Qué módulos y qué contrato toca
La decisión misma	¿Qué los haría cambiar de opinión?	El ADR y el contexto que dio por cierto
La evidencia	¿Cómo saben que es así?	Algo que se pueda correr

SECCIÓN 10

Una falla elegida en el momento

10

El ejemplo: un servicio y su dependencia



`fault.ts latency 2000`	Toda respuesta tarda 2 s más
`fault.ts slow-third 2000`	Lo mismo, con probabilidad de un tercio
`fault.ts cut`	Nadie escucha: la conexión se rechaza enseguida
`fault.ts blackhole`	Se acepta la conexión y no pasa nada más

Dos fallas, el mismo log

make demo (el servicio espera 500 ms como máximo)

```
== dependency: 127.0.0.1:4601 -> 127.0.0.1:4600 · latencia de 1000 ms en todas las conexiones (downstream)
  12 pedidos: 503 × 12 · p50 503 ms, p95 526 ms, máx 526 ms
  pedidos que llegaron a la dependencia: 12 de 12
  pedidos que tardaron más de 150 ms: 12 de 12
  log del más lento: {"route":"/quotes/7","status":503,"outcome":"dependency_timeout","ms":524}

== dependency: 127.0.0.1:4601 -> 127.0.0.1:4600 · corte (proxy deshabilitado: conexión rechazada)
  12 pedidos: 503 × 12 · p50 3 ms, p95 15 ms, máx 15 ms
  pedidos que llegaron a la dependencia: 0 de 12
  pedidos que tardaron más de 150 ms: 0 de 12
  log del más lento: {"route":"/quotes/7","status":503,"outcome":"dependency_unavailable","ms":12}

== dependency: 127.0.0.1:4601 -> 127.0.0.1:4600 · no entrega nada (upstream)
  12 pedidos: 503 × 12 · p50 503 ms, p95 506 ms, máx 506 ms
  pedidos que llegaron a la dependencia: 0 de 12
  pedidos que tardaron más de 150 ms: 12 de 12
  log del más lento: {"route":"/quotes/7","status":503,"outcome":"dependency_timeout","ms":503}
```

Medido en el contenedor del curso, en una laptop con Apple Silicon. Lo que se repite entre corridas es la forma de cada bloque.

Con un timeout, quien llama no sabe si el trabajo se hizo

Falla	El servicio responde	Su log dice	¿Llegó a la dependencia?
Latencia alta	503, al timeout	`dependency_timeout`	Sí, las 12 veces
Blackhole	503, al timeout	`dependency_timeout`	No
Corte	503, enseguida	`dependency_unavailable`	No

Semanas 5 y 12

Un timeout no cancela nada. Si la dependencia cobrara, habría doce cobros y doce compradores a los que se les dijo que falló. Un retry después de un timeout solo es seguro si la operación es idempotente.

«Una de cada tres» es por conexión

make demo

```
== dependency: 127.0.0.1:4601 -> 127.0.0.1:4600 · latencia de 200 ms en 33  % de las conexiones (downstream)
  30 pedidos: 200 × 30 · p50 7 ms, p95 205 ms, máx 207 ms
  pedidos que llegaron a la dependencia: 30 de 30
  pedidos que tardaron más de 150 ms: 2 de 30
  log del más lento: {"route":"/quotes/7","status":200,"outcome":"ok","ms":205}

== la misma falla, por conexión: 8 conexiones al proxy, 6 pedidos por cada una
  conexión 1:    1    1    2    1    1    1  ms
  conexión 2:    3    3    1    0    1    3  ms
  conexión 3:  207  210  203  205  202  202  ms
  conexión 4:  211  203  202  205  210  203  ms
  conexión 5:    2    1    3    1   17    2  ms
  conexión 6:    8    4    2    1    1    1  ms
  conexión 7:    1    1    1    1    1    1  ms
  conexión 8:    2    3    1    1    1    0  ms
```

- Toxiproxy sortea por conexión: una conexión es lenta en todos sus pedidos o en ninguno. Un cliente que reutiliza conexiones no ve «un pedido lento de cada tres».

Con su sistema

terminal

```
svc start toxiproxy
export FAULT_PROXY=gateway              # el nombre que quieran
export FAULT_LISTEN=127.0.0.1:4001      # adonde va a llamar su sistema
export FAULT_UPSTREAM=127.0.0.1:4000    # donde escucha la dependencia real
node /work/code/week-16/fault.ts setup
```

- **Empezar y terminar en `clear`**: el sistema sano, la falla, la recuperación.
- **Carga mientras dura la falla**: sin tráfico no se ve nada.
- **Dos ventanas a la vista**: los logs y la métrica que eligieron.
- Cada comando imprime qué falla quedó puesta. Léanla en voz alta: es la mitad de la predicción.

SECCIÓN 11

La defensa, paso a paso

11

Laboratorio: defensa oral del proyecto

1. **Una compra, sobre el diagrama de contenedores**: qué interviene, qué pasa si falla, dónde queda registrado.
2. **Un ADR al azar**: el contexto, cada alternativa descartada, y qué tendría que cambiar para revisarlo.
3. **`make fitness`**: qué decisión protege cada comprobación. Romper una y ver qué reporta.
4. **Una falla elegida en el momento**: aplicarla y explicarla con sus logs y sus métricas.
5. **Un requisito nuevo**: qué aguanta, qué no y qué decisión se revisa.

Cualquier integrante puede recibir cualquier pregunta

No se repartan la defensa. El ensayo que sirve es el contrario: cada quien explica la parte que menos conoce.

Para cerrar

- Esta semana se entrega **P5 · Cuando algo falla**, con el documento de arquitectura final.
- Antes de la presentación: el guion con `check-doc.ts presentation` y la lista de `defence-checklist.md`.
- Antes de la defensa: las cuatro fallas de `fault.ts` contra su sistema, con carga, anotando qué esperaban y qué vieron.

> **Pregunta**
>
> Elijan una decisión de su Tinkuy. ¿Qué cambio en la carga, en el equipo o en el negocio los haría revisarla?

Lecturas de hoy: Craver, *Stack Overflow: The Architecture — 2016 Edition*; Ingram, *How Discord Stores Trillions of Messages* (2023); Steele, *How Figma's databases team lived to tell the scale* (2024); AOSA vol. 2, *nginx*; Westeinde, *Deconstructing the Monolith* (2019); *Software Architecture in Practice*, cap. 24 y 25; Fowler, *Who Needs an Architect?*