

UNIVERSIDAD NACIONAL DE INGENIERÍA

Arquitectura de Software

Qué es la arquitectura y qué la determina

Estructuras y decisiones que cuesta cambiar · Requisitos significativos, restricciones y stakeholders · C4

Gerson Garrido

Semana 1

Hoy

Sesión A

1. El curso en una página
2. Qué es la arquitectura
3. Tres tipos de estructuras
4. Diseño, funcionalidad y por qué importa

Sesión B

5. Qué determina una arquitectura
6. Dibujar lo que hay: C4
7. Laboratorio

Lecturas

A: Bass, Clements y Kazman, *Software Architecture in Practice* (4.ª ed.), cap. 1 y 2; Fowler, *Who Needs an Architect?* (2003).

B: *Software Architecture in Practice*, cap. 19; Brown, *C4: diagramas* (contexto y contenedores).

Complementarias: Fowler, *Software Architecture Guide*; Brown, *C4: abstractions*; cap. 24 del libro; Kleppmann y Riccomini, *Designing Data-Intensive Applications* (2.ª ed.), cap. 1.

Preguntas que no se responden mirando una función

- ¿Qué pasa con la venta si el servicio que cobra las tarjetas tarda diez segundos?
- ¿Cuántos archivos hay que tocar para aceptar un medio de pago nuevo?
- ¿Aguanta el sistema el minuto en que se abre la venta de un concierto?

De esto trata el curso

Se responden mirando cómo está dividido el sistema, qué partes existen cuando corre, cómo se hablan y dónde viven los datos.

SECCIÓN 01

El curso en una página

01

Cuatro partes y un sistema

1. **Fundamentos y atributos de calidad** (semanas 1–3)
2. **Estilos, tácticas y decisiones** (4–7)
3. **Datos y sistemas distribuidos** (9–13)
4. **Fallas, operación y evolución** (14–16)

Tinkuy

Venta de entradas para conciertos y partidos. Empieza como un monolito pequeño que funciona.

En el proyecto se rediseña seis veces: cada entrega trae un requisito de calidad que el diseño de ese momento no cumple.

La regla de trabajo de todo el semestre

Una decisión de arquitectura:

1. se justifica con un escenario, con su medida;
2. se compara con una alternativa;
3. se escribe;
4. se mide, antes y después, en la misma máquina.

Lo que la regla descarta

"Así lo hace tal empresa" y "esto es más escalable". Un adjetivo sin escenario no se puede discutir: no se sabe cuándo es falso.

SECCIÓN 02

Qué es la arquitectura

02

Tinkuy funciona

tinkuy/

```
$ make db
Semilla 7: 6 locales, 24 eventos, 75 secciones, 13080 asientos (3225 vendidos), 1303 órdenes, 1303 pagos, 3225 entradas, 1303 notificaciones
```

tinkuy/

```
$ make test
...
i tests 62
i suites 9
i pass 62
i fail 0
```

- Las pruebas dicen que el sistema hace lo que debe.
- No dicen qué pasa con doscientas compras a la vez, cuánto cuesta un medio de pago nuevo ni qué ve el comprador si la pasarela se cae.

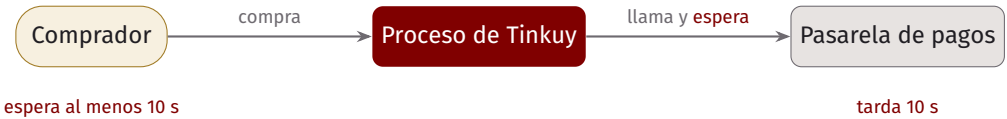
Estructuras para razonar

Arquitectura (Bass, Clements y Kazman)

El conjunto de estructuras necesarias para razonar sobre el sistema. Cada estructura tiene elementos, relaciones entre ellos y propiedades de ambos.

- **Estructuras**, en plural: los archivos, los procesos y las máquinas son tres formas del mismo sistema.
- **Razonar**: responder una pregunta sin leer todo el código ni ponerlo a correr.
- **Propiedades**: cajas y flechas sin propiedades decoran, no sirven para razonar.

Razonar con un elemento, una relación y una propiedad



Sin leer ninguna función

Bastó saber que la llamada es síncrona. Eso es una estructura de arquitectura: sirve para razonar.

Tres consecuencias de la definición

1. **Todo sistema tiene arquitectura**, aunque nadie la haya escrito.
2. **Es una abstracción**: describirla es elegir qué se omite.
3. **No toda estructura es arquitectura**: solo la que ayuda a razonar sobre un atributo de calidad *(quality attribute)* que importa.

Error común

"Es un sistema chico, no tiene arquitectura." La tiene; lo que le falta es alguien que la haya decidido. Y sus consecuencias existen igual.

Las decisiones que cuesta cambiar

Fowler, *Who Needs an Architect?* (2003), citando a Ralph Johnson:

- No son “las decisiones que hay que tomar al comienzo”: son las que uno quisiera acertar al comienzo.
- *“Architecture is about the important stuff. Whatever that is.”*
- Otra forma, del mismo artículo: lo que la gente percibe como difícil de cambiar.

La prueba práctica

¿Cuántos archivos, cuántos equipos y cuántos despliegues toca deshacerla?

Depende del sistema: la base de datos es arquitectura en una aplicación empresarial y puede no serlo en una de imágenes médicas.

La prueba, contada en Tinkuy

code/week-01

```
$ make change-cost
El puerto por omisión es 3000                1 de 31 archivos
La comisión la calcula paymentFeeCents       2 de 31 archivos
Los montos son céntimos enteros             12 de 31 archivos
Las consultas son SQL para PostgreSQL        10 de 31 archivos
```

- Las dos primeras se deshacen en un minuto: no son arquitectura.
- Los céntimos están además en el esquema, en la API y en el protocolo con la pasarela.
- El SQL se lanza desde cinco carpetas: cambiar de base de datos no es reescribir un módulo.
- La medida es gruesa, pero alcanza para ordenar las decisiones por costo.

Dónde deja rastro cada decisión

EN CUÁNTOS DE LOS 31 ARCHIVOS DE SRC/ DEJA RASTRO CADA DECISIÓN

El puerto por omisión es 3000 — 1 de 31 archivos



Solo config.ts, y se cambia con una variable de entorno. No es arquitectura.

La comisión la calcula paymentFeeCents — 2 de 31 archivos



Donde se define y donde se usa: un editor lo renombra en un segundo. Tampoco.

baratas de cambiar arriba, caras abajo

Los montos son céntimos enteros — 12 de 31 archivos



Y fuera de src/: columnas de la base de datos, campos de la API y el protocolo con la pasarela.

Las consultas son SQL para PostgreSQL — 10 de 31 archivos



db.ts y nueve más, repartidos en cinco carpetas: un tercio de los archivos del sistema.

La decisión más grande no está escrita en ningún archivo

> **En Tinkuy**
> “Todo corre en un proceso con una base de datos PostgreSQL.”

Lo que da

- Una compra es una sola transacción
- Un solo programa que desplegar
- Un solo lugar donde buscar un error

Lo que cuesta

- Todo escala junto
- Todo falla junto
- Todo se despliega junto

Cara de cambiar no quiere decir mala. Si conviene depende de lo que se le pida al sistema.

SECCIÓN 03

Tres tipos de estructuras

03

Tres familias, tres preguntas

Estructura de…	Elementos	Pregunta que responde
Módulos	Unidades de código: carpetas, paquetes, clases	¿Qué toca un cambio? ¿Quién es owner de qué?
Componentes y conectores	Lo que existe mientras corre: procesos, bases de datos, colas	¿Qué pasa con la latencia y con las fallas?
Asignación	Software y algo que no lo es: máquinas, repositorios, equipos	¿Quién lo construye y dónde corre?

Módulos: se leen en los archivos

tinkuy/

```
$ grep -rhoE "from '\.\./[a-z]+" src \
    | sort | uniq -c
      3 from '../catalog
      2 from '../config
      9 from '../db
      9 from '../errors
      4 from '../log
      1 from '../notifications
      3 from '../orders
      5 from '../payments
      1 from '../reports
```

- Cinco módulos del negocio, la carpeta `http` y cinco archivos sueltos.
- El `grep` dice qué se importa.
- No dice quién lo importa: `-h` esconde el archivo.
- Para dibujar el grafo hacen falta las dos puntas de cada flecha.

Módulos: quién importa a quién

code/week-01

```
$ make imports | grep -E "/ +-> [a-z]+/"
catalog/       -> orders/             1
http/          -> catalog/            1
http/          -> orders/             1
http/          -> reports/            1
orders/        -> catalog/            1
orders/        -> notifications/      1
orders/        -> payments/           5
reports/       -> catalog/            1
reports/       -> orders/             1
```

- Un cambio en `payments/` puede afectar a `orders/` y a nadie más.
- `catalog/` y `orders/` se importan mutuamente: ninguno se entiende sin el otro (semana 3).
- Lo que no dice: si `orders/ -> payments/` tarda microsegundos o segundos.

Componentes y conectores: se observan con el sistema encendido

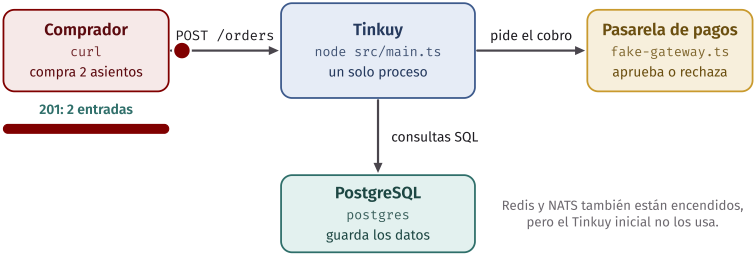
contenedor del curso

```
$ ps -eo pid,args | grep -E "node|postgres$|redis|nats" | grep -v grep
   12 postgres
   80 redis-server *:6379
  104 nats-server --jetstream --store_dir /var/lib/nats --http_port 8222
  193 node tools/fake-gateway.ts
  194 node src/main.ts
```

- Todo Tinkuy es un solo proceso: `node src/main.ts`. Redis y NATS están encendidos, pero el Tinkuy inicial no los usa.
- Si ese proceso se cae, se caen el catálogo, las compras y los reportes.
- Si la pasarela tarda, ¿quién espera? Depende del conector.

Una compra, vista con componentes y conectores

6. Responde 201 con las entradas. El comprador esperó todo lo que tardaron los demás.



TERMINAL DEL SERVIDOR

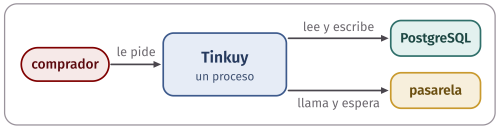
```
pasarela: cobro ch_00000001 approved para orden-1304
correo: confirmación de la orden 1304 enviada a rosa.quispe@correo.pe
orden 1304: pagada, 2 entradas
POST /orders 201
```

El mismo Tinkuy, tres dibujos

De módulos
se lee en los archivos
¿Qué toca un cambio?



De componentes y conectores
se observa con el sistema corriendo
¿Qué pasa si esto falla o tarda?



De asignación
el software y lo que no es software
¿Quién lo construye y dónde corre?



Las tres descripciones son verdaderas. Cuando alguien diga “el componente de pagos”, pregunten: ¿el código o lo que corre?

Asignación: el software y lo que no es software

- En qué máquina corre cada proceso, en qué repositorio vive cada módulo, qué equipo lo mantiene.
- En el curso es mínima: un contenedor de Docker, un repositorio, una pareja.
- Es la que más cambia cuando el sistema crece (ley de Conway, semana 15).

En la industria: Stack Overflow, 2016

4 servidores de SQL Server, 11 web con IIS, 2 de Redis, 3 del motor de etiquetas, 3 de Elasticsearch, 4 load balancers HAProxy.

Dice qué corre dónde. No dice cómo está dividido el código.

Nick Craver, *Stack Overflow: The Architecture – 2016 Edition*, nickcraver.com, 17 de febrero de 2016.

¿Qué estructura hay que mirar?

Pregunta sobre Tinkuy	Estructura
Si cambia el formato de la respuesta de la pasarela, ¿qué archivos reviso?	Módulos
Si PostgreSQL deja de responder, ¿qué ve el comprador?	Componentes y conectores
¿Puedo desplegar un cambio en reportes sin reiniciar las compras?	Componentes y conectores, y asignación
¿Quién aprueba un cambio en el módulo de pagos?	Asignación

Antes de responder, decidan qué estructura mirar. No son niveles de detalle de un mismo dibujo: son dibujos distintos.

SECCIÓN 04

Diseño, funcionalidad y por qué importa

04

Arquitectura y diseño

- No hay una frontera nítida: es la misma actividad a distinta escala.
- Lo que cambia es el costo de equivocarse.
- No existe una lista de “temas que son arquitectura”: `totalCents` es un nombre, pero está en una API que otros ya usan.

Consecuencia práctica

Las decisiones baratas se toman y, si salen mal, se corrigen. Las alternativas escritas y las mediciones se reservan para las caras.

La misma funcionalidad, distintas arquitecturas

	Un proceso, una base de datos	**Varios servicios por la red**
Una compra es una sola transacción	Sí	No: hay que coordinar
Desplegar pagos sin tocar el resto	No	Sí
Una falla en reportes tumba las compras	Puede	No necesariamente
Seguir un error de punta a punta	Un log, un proceso	Varios logs y procesos
Lo que hay que operar	Un programa	Varios, y la red entre ellos

El comprador ve lo mismo y las pruebas funcionales pasan igual. Ninguna columna gana en todas las filas: deciden los atributos de calidad.

Por qué importa la arquitectura

1. Habilita o impide los atributos de calidad. No los garantiza, pero puede hacerlos imposibles.
2. Permite razonar sobre el costo de un cambio antes de hacerlo.
3. Es lo primero que se decide y lo más difícil de corregir.
4. Define cómo se reparte el trabajo entre equipos.
5. Se puede evaluar antes de construir (semana 7).

Dos errores comunes

“Nuestra arquitectura es React y PostgreSQL”

No dice qué partes hay, cómo se comunican ni qué pasa cuando una falla. La tecnología es *una decisión* entre muchas.

“La arquitectura es este diagrama”

El diagrama es *una vista*. La arquitectura es una propiedad del sistema; el diagrama puede estar incompleto o equivocado.

SECCIÓN 05

Qué determina una arquitectura

05

Requisitos significativos para la arquitectura

ASR ***(architecturally significant requirement)***

Un requisito que, si cambiara, cambiaría la arquitectura.

"El comprador puede elegir su asiento."

Si mañana se asignan solos: una pantalla y una función. No es un ASR.

"Cuando se abre la venta de un concierto, 20 000 personas intentan comprar en el primer minuto."

Con 20 personas, cualquier diseño sirve. Determina la arquitectura.

Dónde buscar un ASR

Casi nunca dicen *qué hace* el sistema. Dicen:

- cuánto: cuántos usuarios, cuántos datos;
- qué tan rápido;
- con qué frecuencia cambia algo;
- qué pasa si falla.

> **Casi nunca no es nunca**
>
> “El lector recibe un aviso un día antes del vencimiento” es funcional, pero obliga a que exista algo que corra solo cada noche.

Un deseo todavía no es un requisito

Error común

“Debe ser escalable.” “Debe ser seguro.” “Debe ser rápido.”

- ¿Escalable hasta cuánto? ¿Rápido para quién, haciendo qué?
- Si no se puede decir cuándo se incumple, todavía no es un requisito.
- Esta semana los encontramos. La próxima los convertimos en escenarios con medida.

De dónde salen

1. **Los requisitos escritos.** Pocas veces: suelen decir “debe ser rápido”. Cada frase vaga marca dónde preguntar.
2. **Los stakeholders.** Preguntas con situación: “¿qué pasó la última vez que se abrió la venta de un concierto grande?”
3. **Los objetivos del negocio.** “Vender las entradas de los partidos de la selección” trae adentro una carga que el sistema nunca recibió.
4. **Lo que se sabe del dominio.** Nadie escribe “un asiento no se vende dos veces”.

Hacerlos explícitos es parte del trabajo de arquitectura.

En la industria: una preventa, noviembre de 2022

Lo que dice el comunicado de Ticketmaster:

- Más de 3.5 millones de personas inscritas.
- Alrededor de 1.5 millones recibieron un código.
- El día de la venta: 3 500 millones de pedidos, cuatro veces el pico anterior.
- Cerca del 15 % de las interacciones tuvo problemas.

Lo que leemos nosotros

Se planificó para los invitados. Llegaron los invitados, los que no tenían código y los bots.

El ASR no es el día promedio: es el pico, y el pico incluye a quien nadie invitó.

Ticketmaster, *Taylor Swift | The Eras Tour Onsale Explained*, business.ticketmaster.com, 19 de noviembre de 2022.

Stakeholders: no solo quien usa el sistema

En Tinkuy	Qué quiere del sistema
Compradores	Compra rápida y, si pagaron, que la entrada exista
Organizadores	Vender todo el aforo y ver sus ventas al momento
Soporte	Explicar qué pasó con una orden
Quien opera de madrugada	Enterarse antes que los compradores
El equipo que lo mantiene	Cambiar una parte sin romper otra
La pasarela de pagos	Pedidos bien formados, y no la misma orden cobrada dos veces
El área legal	Que los datos se traten como dice la ley

Piden atributos distintos y a veces opuestos: el reporte al momento y la compra rápida usan la misma base de datos.

Restricciones: lo que ya viene decidido

- **Técnicas:** el lenguaje que el equipo conoce, la base de datos que la organización ya opera.
- **Del negocio:** el presupuesto, el plazo, la pasarela con la que ya se firmó.
- **Legales:** la Ley de Protección de Datos Personales (Ley N.° 29733).

Requisito o restricción

Un requisito se negocia. Una restricción no: se diseña dentro de ella.

"Tiene que ser con microservicios" casi nunca es una restricción: es una solución propuesta antes de tiempo.

Lo que también influye

- **El equipo**: dos personas no operan doce servicios.
- **El plazo**: con tres meses se elige distinto que con tres años, y está bien si se escribe.
- **Lo que el equipo construyó antes**: lo conocido tiene menos sorpresas, y es un sesgo.

> **En los dos sentidos**
>
> La arquitectura también moldea a la organización: los módulos se vuelven equipos. Lo retomamos con la ley de Conway (semana 15).

Un ejemplo completo: Qillqa, préstamos de una biblioteca

1. El lector busca libros por título, autor o materia.
2. Hasta tres libros a la vez, por siete días, con dos renovaciones.
3. Solo los estudiantes matriculados pueden pedir prestado.
4. El lector recibe un aviso un día antes de que venza su préstamo.
5. El sistema debe ser rápido y seguro.
6. El primer día de parciales, a las 8 a. m., hay unas doscientas personas en el mostrador.
7. Las reglas de préstamo las cambia el comité cada semestre.
8. Un préstamo registrado en el mostrador no se puede perder.

¿Cuáles cambiarían la arquitectura si cambiaran?

Qillqa: la prueba del cambio

#	¿ASR?	Por qué
1	No	Una consulta
2	No, tal como está	Tres números en una función (pero véase el 7)
3	Esconde uno	Depende de un sistema ajeno: ¿y si no responde?
4	Sí	Hace falta algo que corra solo cada noche
5	Todavía no es un requisito	No se puede decir cuándo se incumple
6	Sí	Cuánto y cuándo: un pico en el mostrador
7	Sí	Con qué frecuencia cambia
8	Sí	Qué pasa si falla

Qillqa: un conflicto y una decisión

- Los bibliotecarios: el mostrador nunca se detiene.
- La oficina de registro: no nos llamen por cada préstamo.
- La regla 3: hay que verificar la matrícula.

No se cumplen las tres al máximo.

Una decisión

Guardar una copia de la lista de matriculados y aceptar que esté desactualizada unas horas.

Alternativa: consultar siempre y no prestar sin respuesta.
Costo: por unas horas, alguien que se retiró todavía puede llevarse un libro.

Tinkuy, primera lectura

La lista de ASR sale de la clase y del laboratorio. Dónde buscar:

- **¿Cuánto?** El evento 1 tiene 2000 asientos. ¿Cuánta gente llega cuando abre su venta?
- **¿Qué tan rápido?** ¿Es lo mismo esperar al mirar el catálogo que al pagar?
- **¿Con qué frecuencia cambia?** Hoy hay dos medios de pago. ¿Cada cuánto aparece otro?
- **¿Qué pasa si falla?** Para cada pieza: si se cae o tarda, ¿qué ve el comprador?

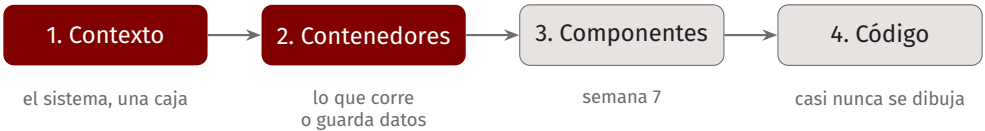
La próxima semana, cada candidato se vuelve un escenario medible.

SECCIÓN 06

Dibujar lo que hay: C4

06

C4: cuatro niveles de zoom



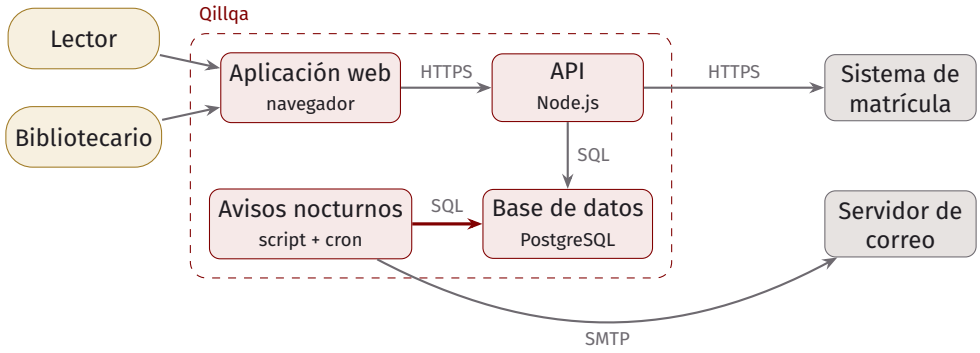
- Hoy, los dos primeros. Según el propio modelo, bastan para la mayoría de los equipos.
- El de contenedores es una vista de componentes y conectores a grano grueso.

Contexto: el sistema como una caja



- Fija el límite, quién lo usa y de quién depende. Adentro, nada: ni bases de datos ni procesos.
- Una flecha es un *envío*; la otra, una *pregunta* en medio de un préstamo. ¿Y si no responde?

Contenedores: la caja, abierta



- Cuatro piezas y solo dos son servidores. El requisito 4 se volvió una caja.
- Decisión a la vista: los avisos leen la base de datos sin pasar por la API. Más simple, y dos programas conocen las tablas.

Un contenedor de C4 no es un contenedor de Docker

Contenedor (C4)

Una aplicación o un almacén de datos: algo que tiene que estar corriendo, o existir, para que el sistema completo funcione.

- Un servidor, una aplicación en el navegador, una base de datos, un programa que corre cada noche.
- En el curso todo corre dentro de un solo contenedor de Docker, y el diagrama de contenedores tiene varias cajas.
- Docker es despliegue: estructura de asignación.

Simon Brown, c4model.com/abstractions/container (la página empieza con “Not Docker!”).

Diagramas como texto: dos tablas

docs/c4/context.md

```
# Qillqa: diagrama de contexto
- Nivel: contexto
- Sistema: Qillqa

## Elementos
| Nombre | Tipo    | Descripción                                  | Tecnología           |
|--------|---------|----------------------------------------------|----------------------|
| Lector | persona | Busca libros, los reserva y renueva préstamos | -                    |
| Qillqa | sistema | Catálogo, reservas y préstamos               | Node.js y PostgreSQL |

## Relaciones
| De     | Hacia  | Qué viaja                              | Protocolo                |
|--------|--------|----------------------------------------|--------------------------|
| Lector | Qillqa | Busca libros, reserva y renueva        | HTTPS, desde el navegador |
```

Extracto abreviado. El formato completo y sus reglas: `code/week-01/c4-format.md`.

Lo que el formato exige y un dibujo deja pasar

1. Toda caja dice qué es y de qué es responsable. “Base de datos” es el tipo, no la descripción.
2. Toda caja dice con qué tecnología está hecha (menos las personas).
3. Toda flecha dice qué viaja, con un verbo. “Usa” no dice nada.
4. Toda flecha dice por qué protocolo: HTTP síncrono y un mensaje en una cola fallan distinto.

La flecha va en la dirección de quien inicia. En el curso se pide el protocolo también en el contexto, más de lo que C4 recomienda para ese nivel.

Un diagrama con errores

code/week-01 (extracto de make errors)

```
errors/containers.md:16: [tecnologia] "Backend" no dice con qué tecnología está hecho
errors/containers.md:17: [suelto] "Docker" no participa en ninguna relación: o falta una flecha o sobra la caja
...
errors/containers.md:27: [protocolo] Frontend → Backend no dice por qué protocolo
...
errors/containers.md:29: [extremos] "Sistema de matrícula" no está en la tabla de Elementos
errors/containers.md: contenedores de Qillqa, 7 elementos, 4 relaciones: 12 errores
```

- Celdas vacías: en un dibujo pasan inadvertidas; en una tabla se ven.
- “Docker” como contenedor: lo delata que no se relaciona con nadie.
- Una flecha hacia una caja que no existe y una caja a la que nadie le envía nada.

Completo no es lo mismo que cierto

- Al diagrama con errores le faltan los avisos nocturnos. El revisor no lo dijo: no tiene cómo saberlo.
- Los contenedores se comprueban contra `ps`.
- Las flechas, contra el código que hace la llamada y los logs.
- Los sistemas externos, contra la configuración: cada URL apunta a algo.

Un diagrama es una afirmación

Puede ser falsa. Un diagrama completo y falso es peor que no tener diagrama, porque se le cree.

¿Por qué tablas?

	Se gana	Se paga
Herramienta gráfica	El mejor dibujo	No se compara línea por línea; se desactualiza en silencio
Lenguaje de diagramas (Mermaid)	Se versiona y se dibuja solo	Descripción, tecnología y protocolo son opcionales
Tablas (el curso)	Se versiona, se revisa y un programa comprueba que no falte nada	No se ve el dibujo

Elegimos las tablas porque lo que más se olvida en un diagrama es lo que más sirve para razonar.

SECCIÓN 07

Laboratorio

07

Laboratorio: el entorno, Tinkuy y sus primeros diagramas

En parejas, sobre una copia de Tinkuy en su repositorio. Hoy no se cambia ni una línea.

1. `./env/run.sh` y `svc status`: PostgreSQL, Redis y NATS arriba.
2. `make db`, `make gateway`, `make start`; probar los endpoints con `curl`.
3. `make test` y `make typecheck`. Leer una prueba: ¿qué comportamiento fija?
4. Módulos: una frase por carpeta, contar los imports y dibujar el grafo.
5. Componentes y conectores: `ps aux`, puertos, y seguir `POST /orders` de punta a punta: qué llamadas salen del proceso y cuáles van dentro de la transacción.
6. `docs/c4/context.md` y `docs/c4/containers.md`, y pasarlos por `check-c4.ts`.
7. Tres candidatos a ASR y qué parte del diseño actual no los aguantaría.

> **Dibujen el sistema como es hoy**
>
> Con lo que observaron, no con lo que suponen ni con lo que creen que debería ser.

Para la próxima clase

- Terminar el laboratorio. Se publica **P0 · Reconocimiento**, individual: los diagramas de hoy son su primera parte.
- Leer: *Software Architecture in Practice*, cap. 1, 2 y 19; Fowler, *Who Needs an Architect?*; Brown, *C4: diagramas*.
- Semana 2: atributos de calidad y escenarios. Los tres candidatos a ASR de hoy se vuelven escenarios con medida, y se miden.

Pregunta

En el último sistema que construyeron, ¿cuál fue la decisión más cara de cambiar? ¿Alguien la decidió, o salió sola?