Status call

DDD, 20.11.2026, 18:00

Updated after the call: state of the repos 20.11.2026 19:15

Agenda (30 minutes)

- Overview: bounded context teams
- Overview: specialist teams
- Across teams: the contracts one week later
  - what got better
  - the chain of events
  - who owns it? (part 2)
  - valid again, but nobody checks
- The ADRs
- ADR 0007, recipes for the kitchen: the consequences
- AI tactical: one entry point for coding agents
- Decided in the call
- Next steps

Bounded context teams

Team	Contract on main	Aligned with the neighbours	#3 Start implementing	UI issue
order-payment-recipe	complete order (!6) and recipe merged today, with examples	mostly open: the stock event (ingredient)	started skeleton on main; first event flow in !7; no pipeline	#5, no date
foodprocessing-stock	valid again !5 merged today, broken file removed	partly open: `tableId` for distribution, the stock event	skeleton Spring Boot set up today, pipeline failed	#5, no date
distribution-robots	valid again typo fixed today	mostly one delivery per order since the call; still needs `tableId`	done Kafka (!8), event in, event out, with tests (!9); pipeline green	#10, no date
reservation-table	aligned v0.4.0: orders matched by table	done examples for every message	done `order.placed` in, state out; tests up to Kafka; pipeline green	#5, no date

Specialist teams

Team	Done	Open
exploration	`done` repository with case study and EventStorming events (one file per event, plus an index); process for changes; robots in the case study since 19.11.	#14 Check the contracts against the EventStorming result exploration #1: reservation events and the first order of a reservation (!4, approved)
strategic	`done early` user tasks mapped to the contexts, one UI issue per team	#18 format for agents: Miro MCP, yes, as an ADR strategic #1 domain stories: your artifact, update them
tactical	meeting with both sub-teams today: two proposals taken on (UI frame, splitting a context into services)	tactical #5 UIs in the workshops tactical #6 one entry point for coding agents tactical #1 to #4
architecture	`#3` header list in ADR 0004, ownership in ADR 0005 `0006` and `0007` merged three-day comment period in the README	0008: events between services (MR !7) #5 the aggregate table

Issue numbers in `sample-service`, unless a repository is named

Across teams: what got better in one week

Fixed since 13.11.

- all four contracts: the outbox-lib header names
- nobody else publishes on `bl.order.events.v1`
- table ID: a UUID everywhere
- ordering ↔ kitchen: `cooking.started`, `cooking.finished` on `bl.cooking.events.v1`
- ordering ↔ reservation: the reservation events; `order.placed` matched by table
- distribution → ordering: `delivery.food-delivered`
- order and recipe contracts complete, on main
- every contract file on main parses again
- ADR 0004 lists the headers, ADR 0005 says who owns a topic

> This afternoon most of it went to main. Main is what the neighbours build against.

Across teams: the chain of events, still open

Link	Producer says	Consumer says
kitchen → distribution	`cooking.finished`: `orderId`	`orderId`, `tableId`
stock → recipe	kitchen: `id` (integer), `name`, `delta`, `unit`	recipe: `stockItemId`, `ingredientId`, `quantityDelta`, …, `revision`

> The producer's contract is the reference. If the consumer needs more, it asks the producer in an issue or MR.

Guardians: exploration (#14) for the events, architecture for topics and headers (ADR 0004, 0005)

Across teams: who owns it? (part 2)

What we see

- ADR 0005: only the owner publishes on an aggregate's topics
- kitchen: cooking state still sent on `db.order.ecst.v1`
- the recipe names an ingredient by `ingredientId`
- the stock names it by an integer `id` and a `name`
- no shared ID between the two

Clarified in the call

- "ingredient" (recipe) and "stock item" (stock): the same thing, the tomatoes, under two names
- recipe and stock teams agree among themselves who owns it
- the context that owns an aggregate defines its ID
- that ID is used throughout the system

> Which aggregates exist and who owns them follows after the tactical workshop on 27.11. Then ADR 0005 is split, and its table filled.

Across teams: valid again, but nobody checks

Fixed this afternoon

- distribution-robots: first line is `asyncapi:` again
- kitchen: the broken order contract and the two empty files are gone
- all seven contract files on main parse as AsyncAPI

Still the case

- no pipeline validates a contract
- order-payment-recipe: no pipeline at all
- kitchen: pipeline on main failed
- the typos went unnoticed for one and two days

How to fix it

- each service adds one job in the pipeline:
  `npx @asyncapi/cli validate`
- no direct commits to main for contracts

The ADRs

Done

- 0004: header list and a block to copy
- 0005: ownership, with a table of aggregates
- 0006: distribution assigns the robot
- 0007 merged: the kitchen consumes the recipe events (option 2)
- README: three days for comments on every ADR MR

Open

- 0008 proposed: events between services, inside a context too (MR !7)
- 0005 names its own gap: who creates the topics
- 0005 holds two decisions: naming, and ownership; split after 27.11.

> Decided in the call: no status line in the ADRs any more. An open MR is proposed, a merged ADR is accepted. The architecture team removes the status lines.

ADR 0007, recipes for the kitchen: the consequences

Where we stand

- proposal: the order message carries the recipes (option 1)
- ordering objected: coupling, data nobody else needs
- kitchen accepted option 2, to avoid a long discussion
- option 2: the kitchen consumes the recipe events and keeps its own copy
- ADR updated and merged today

Consequences of option 2, still open

- recipe changed or deleted after the order: which version does the kitchen cook?
- order arrives before the recipe: two topics, no order between them
- the kitchen's copy of all recipes: built from which topic, after a restart?
- steps and ingredients of a recipe: whose knowledge is that?
- the recipe's ingredients and the kitchen's stock: no shared ID (part 2)

> Not discussed in the call: too late, and too large a group. We come back to these questions.

AI tactical: one entry point for coding agents

- proposal of the AI tactical team (tactical #6)
- agreed with the specialist teams?

> Where does an agent working on one service find the EventStorming result, the strategic artifacts, the ADRs, the neighbours' contracts and the course material?

Decided in the call

Architecture

- UI: every bounded context provides its own UI, no monolithic UI
- tactical team proposes a simple HTML frame that hosts the contexts' UIs in iframes
- a context may split into several services; between them the same rules as between contexts: events wherever possible (ADR 0008)

Ownership and ADRs

- ingredient = stock item; recipe and stock teams settle the owner
- the owner of an aggregate defines its ID, used throughout the system
- ADRs without a status line
- ADR 0005 split after the tactical workshop
- ADR 0007 consequences: revisited later

Next steps

- order-payment-recipe, foodprocessing-stock: who owns the ingredient; the owner defines the ID
- foodprocessing-stock ↔ distribution-robots: `tableId` in `cooking.finished`
- order-payment-recipe, foodprocessing-stock: #3 one event in, one event out, with a test; a pipeline
- all context teams: a contract validation job in the pipeline
- exploration: #14 check the contracts
- tactical: the UI frame with iframes
- architecture: review ADR 0008 (three days); remove the status lines
- 27.11.: tactical workshop, building blocks and aggregates; then the aggregate table and the split of ADR 0005

Building a service in four steps

STEP 1

Core functionality

What does the service do?

- aggregates with `domain` and `application`
- business rules, persisted with JPA
- no way in, no way out, no tests yet

module only: service-development-1

STEP 2

Service without a broker

How does an event get in and out, testably?

- receiver and handler per incoming event
- an interface for each outgoing event
- tests on three levels, no Kafka

infopage, variant with Spring events

STEP 3

Connecting to Kafka

What changes for Kafka?

- nothing: only adapters are added
- in: Kafka listener with consumer-lib
- out: the interface implemented with outbox-lib (ADR 0004)

infopage

STEP 4

Testing against a real Kafka

Does Kafka really deliver it that way?

- Kafka in a Docker container (Testcontainers)
- the test plays both neighbours
- the service code does not change

infopage

> distribution-robots: steps 3 and 4 on main. reservation-table: steps 2 and 3, with an embedded Kafka in the tests.