Status call

DDD, 13.11.2026, 18:00

State of the repos: 13.11.2026 14:30

Agenda (30 minutes)

- Overview: bounded context teams
- Overview: specialist teams
- Across teams: what the first contracts show
  - who owns the order?
  - the chain of events
  - event headers
  - valid is not the same as right
- ADR 0004 and 0005
- Until 20.11.

> For context: the picture of the teams comes from an AI-assisted review of the repositories, without a thorough manual cross-check. It shows the state of 13.11.2026, 14:30. Anything merged later is missing.

Bounded context teams

Team	#1 Create the repository	#2 Model the events in AsyncAPI	#3 Start implementing	Checkboxes
order-payment-recipe	done	open no AsyncAPI file yet	due 20.11.	used
foodprocessing-stock	done	partly stock complete, order events a skeleton	due 20.11.	used
distribution-robots	done closed in sample-service #3	mostly your #1; examples for both events	due 20.11. your #2	used
reservation-table	mostly draft MR !1 still open	open draft MR !2, no contract yet	due 20.11.	partly

> Every team has its repository, moved its issues there and ticks its checkboxes. That is the state everyone can see.

Specialist teams

Team	Due today	Due 20.11.
exploration	open #13 Create the repository with the EventStorming result: no repository `exploration` yet	#14 Check the contracts against the EventStorming result #17 Format of the EventStorming result for agents (with AI tactical)
strategic	started #15 Create the repository with its artifacts: repository since 12.11., README a placeholder, no artifacts yet	#18 Format of the strategic artifacts for agents (with AI tactical) #19 Check the contracts against the domain stories
tactical	done #16 Create the repository, add the slides of 06.11.; four issues of its own on formats for agents	#20 Propose one entry point for coding agents, presented on 20.11.
architecture	#1 README #2 C4 #3 ADRs level 1 in the repo, level 2 on Miro; ADR 0004, 0005 still proposed	#3 Decide ADR 0004 and 0005 (due today, proposed: 20.11.)

Issue numbers in `sample-service`, for architecture in `architecture`

Across teams: who owns the order?

Context	publishes on `bl.order.events.v1`	its own aggregate
order-payment-recipe	(no contract yet)	the order of the guest
foodprocessing-stock	`order.started-cooking`, `order.finished-cooking`, plus state on `bl.order.ecst.v1`	?
distribution-robots	`delivery.food-delivered`	`Delivery`

Why this is a problem

- ADR 0005: one topic per aggregate
- three producers on one topic: nobody owns its schema
- "order" means something else in each context
- exactly what bounded contexts separate

Questions

- which context owns the order?
- what does the kitchen call the thing it cooks?
- does ADR 0005 need "an aggregate belongs to one context"?

Across teams: the chain of events does not fit yet

Happy path: reservation-table → order-payment-recipe → foodprocessing-stock → distribution-robots

Link	Producer says	Consumer says
reservation → ordering	(no contract yet)	(no contract yet)
ordering → kitchen	(no contract yet)	`order.confirmed`, a placeholder written by the kitchen
kitchen → distribution, name	`order.finished-cooking-provisional`	`order.prepared` (example header: `dish.prepared`)
kitchen → distribution, payload	none	`orderId`, `tableId`

> The producer's contract is the reference. The consumer refers to it, rather than describing the neighbour's event in its own words. That is why the contracts upstream are needed first.

Across teams: two different sets of headers

	Headers
outbox-lib writes (ADR 0004)	`event_id`, `entity`, `sequence_id`, `created_at`, `event_type`, `event_type_version`, `trace_id`
foodprocessing-stock	five of them, with the binary types of the library
distribution-robots	`type`, `eventId`, `transactionId`, `timestamp`: none of them

Why this is a problem

- the contract promises headers that never arrive
- an event can arrive twice (ADR 0004)
- the consumer recognises a duplicate by the header `event_id`
- a consumer that reads `eventId` finds nothing

How to fix it

- the library decides the names: `HeaderKeys` in `outbox-lib-core`
- ADR 0004 lists them once, every contract copies the block

Across teams: valid is not the same as right

What we see

- AsyncAPI validator: all three contract files valid, no warning
- distribution-robots: schema of food-delivered allows only `orderId`, both examples carry three more fields
- foodprocessing-stock: order messages without payload and headers
- foodprocessing-stock: started-cooking and finished-cooking both send `order.confirmed`
- foodprocessing-stock: state topic named `bl.order.ecst.v1`, ADR 0005 says `db.<aggregate>.ecst.v1`
- foodprocessing-stock: an empty `docs/asyncapi.yaml` next to the two contracts

Why this is a problem

- the validator checks the structure, not the meaning
- examples are not checked against the schema
- the neighbours build against the examples

How to fix it

- read your contract as the consumer would
- review by the consuming team, and by exploration (sample-service #14)
- validate in the pipeline anyway: `npx @asyncapi/cli validate`, catches broken references
- later: a test checks the published event against your example

ADR 0004 and 0005: still proposed

What we see

- both contracts already build on them
- no comment in `architecture#3` yet
- today's slides are the missing feedback

What the ADRs do not say yet

- 0005: only the context that owns an aggregate publishes on its topics, and aggregate names are unique across contexts (slide "who owns the order?")
- 0004: which headers outbox-lib writes, so that every contract copies the same block (slide "headers")
- 0005 names this gap itself: who creates the topics in Kafka

> Decided by 20.11., so that the implementation starts on a fixed basis. Feedback goes into `architecture#3`.

Until 20.11.

- order-payment-recipe, reservation-table: #2 Model the events in AsyncAPI, early in the week
- distribution-robots, foodprocessing-stock: topic, headers, examples; agree on the event between you
- all four: #3 Start implementing (distribution-robots: #2): one event in, one event out, a test without a broker
- exploration: #13 repository with the EventStorming result, #14 check the contracts, #17 format for agents
- strategic: #15 repository with the artifacts, #18 format for agents, #19 check against the domain stories
- AI tactical: #20 one entry point for coding agents, presented on 20.11.
- architecture: architecture #3 Decide ADR 0004 and 0005

Context teams: numbers in your own repository. Specialist teams: numbers in `sample-service`.

Where to find how

The contract

- Infopage Event contracts with AsyncAPI, module `asyncapi-event-contracts` in learn-event-driven-architecture
- ADR 0004 and 0005 in the `architecture` repository

#3 Start implementing (distribution-robots: #2), in this order

- Service without a broker
- Spring events instead of a handler
- Connecting a service to Kafka

The headers

- `HeaderKeys` in outbox-lib, and its README

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 COMING SOON

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 and module in preparation

> Your issue #3 (Start implementing) is steps 1 and 2: one event in, one event out, a test without a broker. Example in all four steps: billing in a bike sharing system, repo `learn-event-driven-architecture`.