BEARS-1530 :
Uploader - Handle null values correctly for MarketBettingControlsUpdated.getBetDelay()

Sarath Ramachandran

Table of Contents

Apprenticeship Activity Report

Introduction

The objective of BEARS-1530 was to resolve issues observed in the event processing pipeline of the retail-msk-dynamo-uploader, where event records with missing or null fields were being written to the DynamoDB store.

I collaborated with product owners, platform engineers, and QA testers in an agile team setting. The work was tracked using Jira (under the BEARS-1530 ticket), with regular updates shared in stand-ups and code reviews. Development was version-controlled via GitLab and tested via both manual testing and automated unit test coverage.

This report outlines the full lifecycle of this feature—from understanding the requirements, planning and design, implementation and debugging, to testing and deployment—following standard SDLC practices and reflecting on the impact and learning outcomes achieved through this work.

Summary

This feature addresses a null pointer exception that occurred in the DocDbService class when handling EVENT_BETTING_CONTROLS_UPDATED Kafka messages. The betDelay field in MarketBettingControlsUpdated could sometimes be null, which was not previously handled. The change introduces fallback logic using the betDelayDefaultvalue from the event message when market-level bet delay is missing.

Business Rationale: The live platform was intermittently crashing or logging noisy stack traces when betDelay was missing in messages from the upstream Kafka stream. This disrupted stability and introduced inconsistencies in downstream systems relying on DocDB. The aim was to gracefully fall back to the event-level betDelayDefault value and prevent system failure.

Requirement Gathering

The requirement for this task was documented in Jira under the ticket BEARS-1530.

- Ensure that event creation logic gracefully handles missing/null fields in DynamoDB documents.
- Prevent invalid entries from being committed to the database.
- Collaborate with automation testers to resolve persistent pipeline failures.
- Create a test-safe and QA-safe environment for Competitions filtering with valid null handling logic.

Tools & Development Environment :

- Jira for tracking the BEARS-1530 ticket and subtasks.
- GitLab Merge Request and comments for peer review and feedback.
- **NodeJS**, **Typescript**, **KafkaJS**
- **Mocha** (Unit testing)
- **GitLab CI/CD**
- **MongoDB** (for event simulation in tests)
- **DynamoDB** (production document storage)
- **IntelliJ IDEA**
- Splunk

Stakeholders Involved :

- SETL – clarified intended shared usage of CompetitionFilter.
- QA Engineers – highlighted edge cases.
- Product Owner – defined scope and confirmed visual/functional parity expectations.

We worked in a Kanban-based Agile workflow. The ticket moved through stages :
To Do → In Progress → Code Review → QA → Done, with team input gathered during stand-ups, pair debugging sessions, and MR reviews.

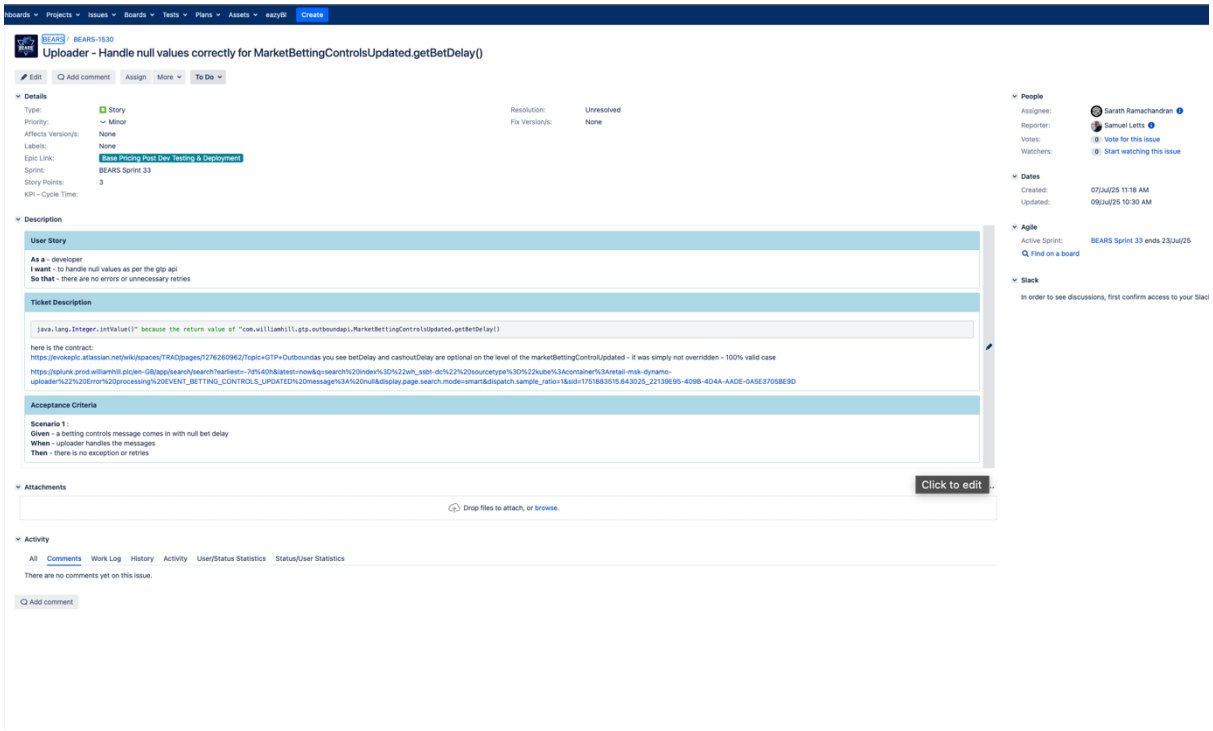

Figure 1 : Story ticket details

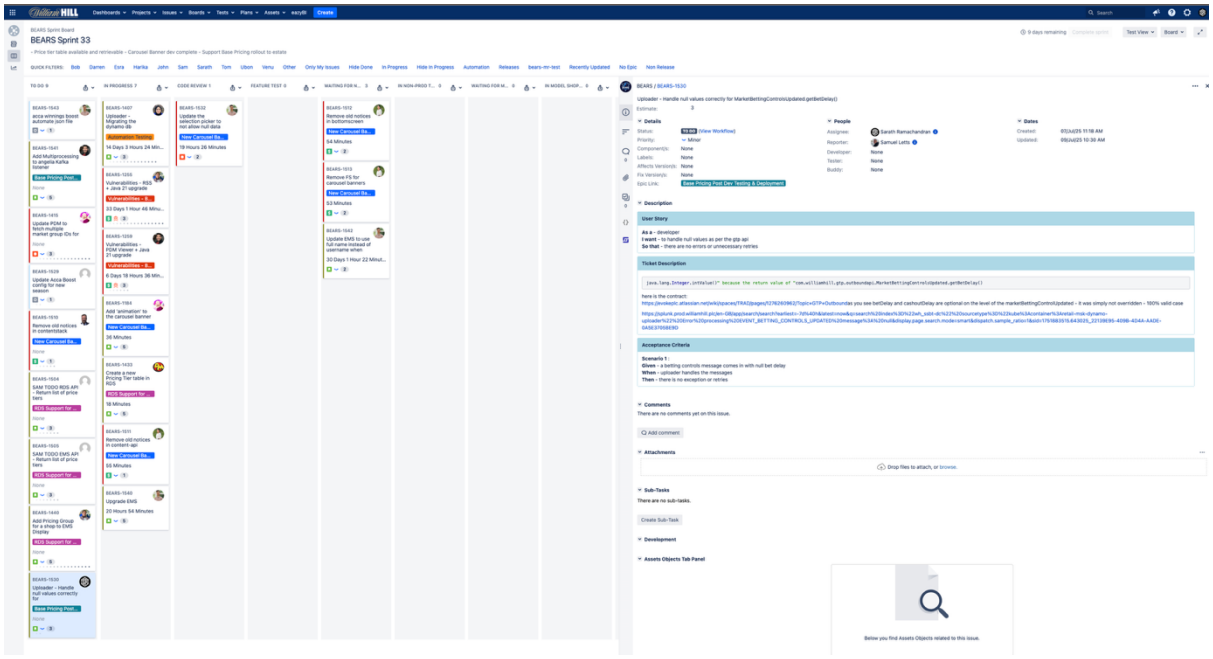

Figure 2 : Sprint board

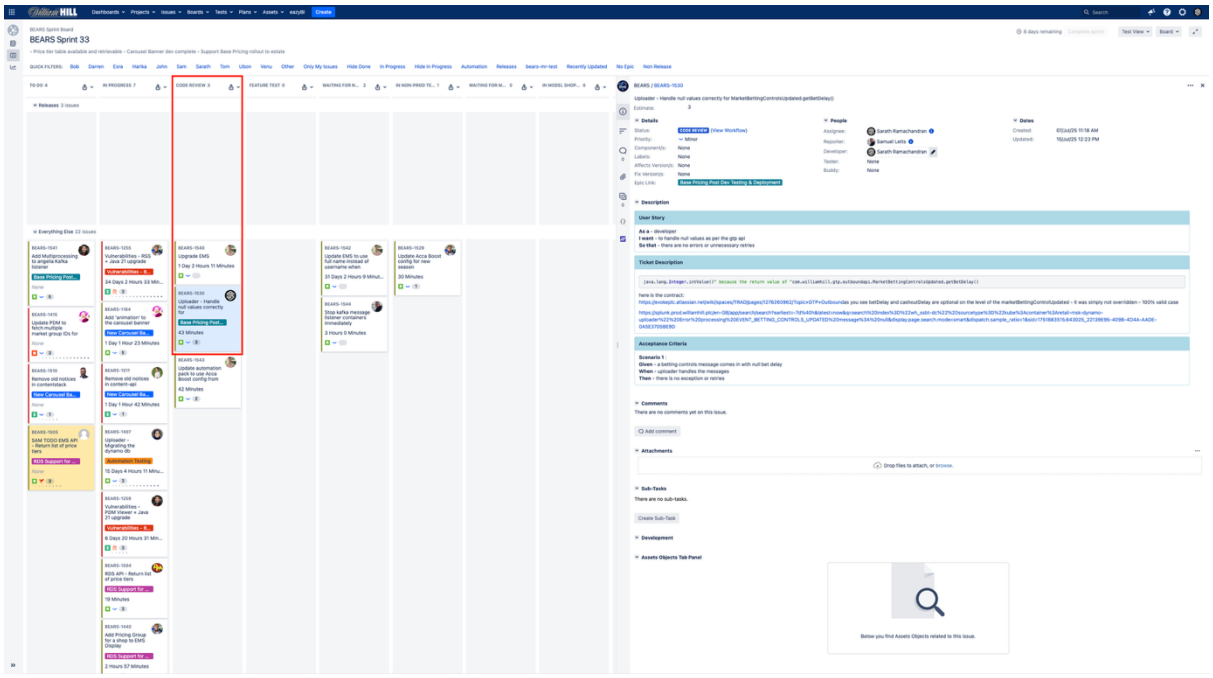

Figure 3 : Sprint board (Ticket in Code Review)

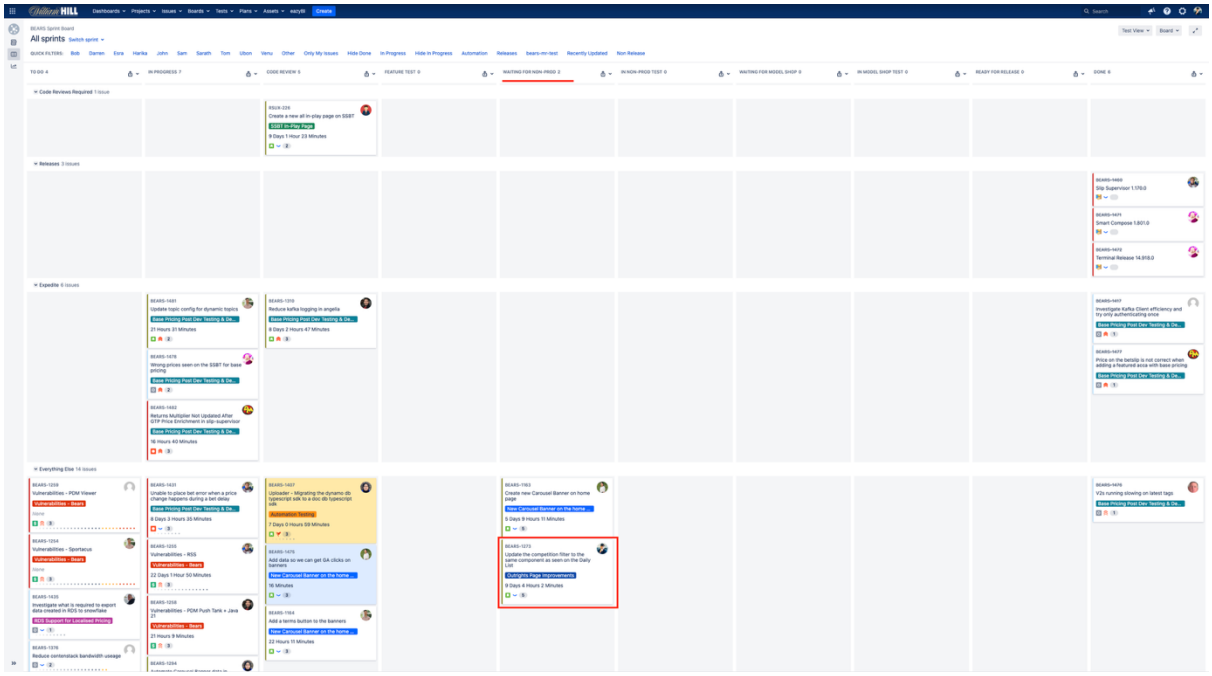

Figure 4 : Ticket in Waiting for non prod stage

Analysis

- Identified core issue where undefined or null values in Kafka messages passed through the uploader pipeline and resulted in incorrect or incomplete records.
- Debug logs revealed that document inserts proceeded even if essential fields (like selection IDs or market keys) were missing.

Scope of work

- Modify DocDbService.java to handle null betDelay by falling back to message.getBetDelayDefault().
- Update associated unit tests to reflect the new logic.
- Add a new test case to validate the fallback behaviour.
- Confirm that existing MarketDoc entries are updated or inserted with the correct delay.
- Validate with Kafka test messages locally and confirm logs.
- Coordinate with automation team to verify integration test behaviour.

Implementation

Code changes

File: DocDbService.java

Added:

```
if (delay == null) {
    delay = message.getBetDelayDefault();
}
```

Removed previously skipped logic for null delays (early return).

Unit Tests Added

- `shouldInsertMarketWithDefaultBetDelayWhenBetDelayIsNull`

Verifies that a `MarketDoc` is inserted using the `betDelayDefault` when the market's `betDelay` is null.

Test Strategy

- Unit tests executed via `mvn verify`
- Local test using Kafka UI with mock message payloads
- Full automation test suite run in GitLab CI pipeline

Validation Details

- Manual Kafka message sent with null market-level delay and valid event-level delay.
- Logs confirmed default delay was applied.
- Document DB entries verified using MongoLens.
- Pipeline passed post change.

Challenges Encountered

- Automation test failures initially due to unrelated configuration.
- Collaborated with automation tester to validate test coverage.
- Fixed IntelliJ environment issues and TypeScript red underline errors during local testing.

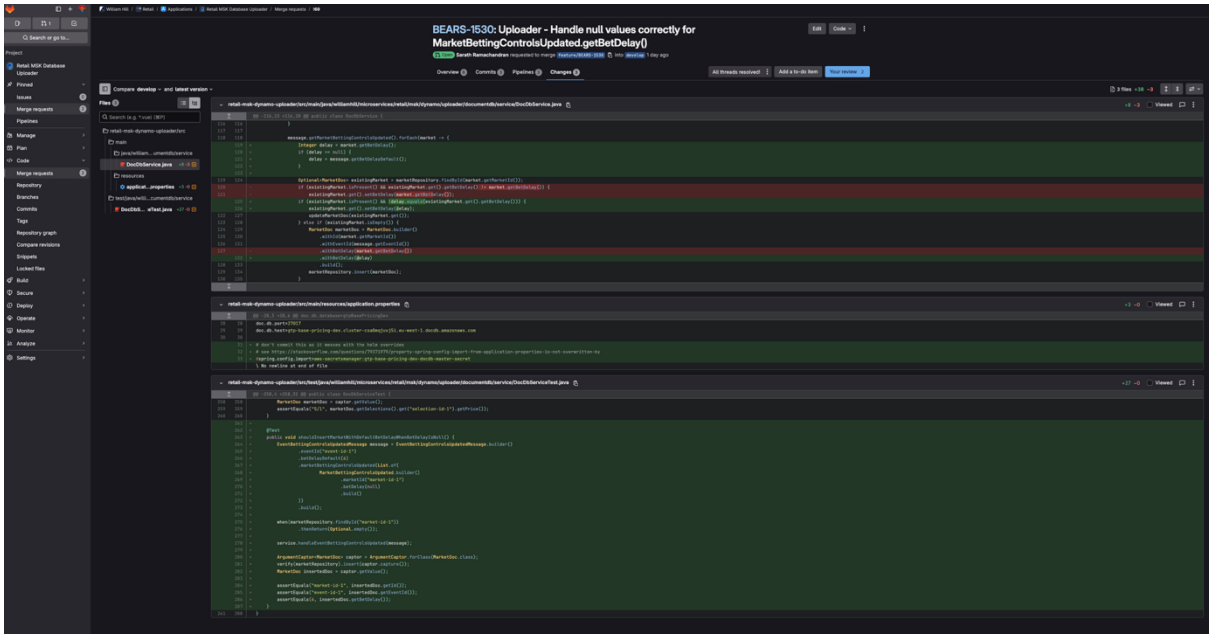

Figure 5 : Code changes

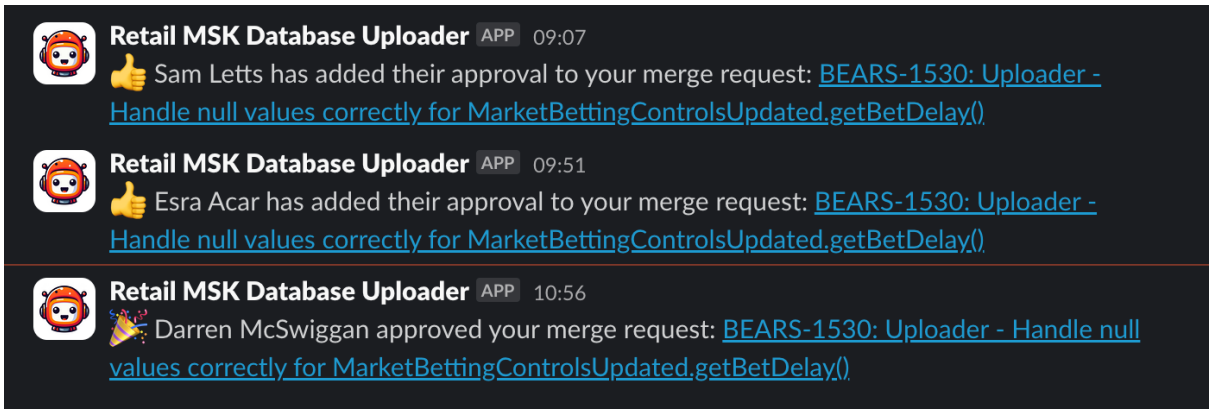

Figure 6 : MR approvals

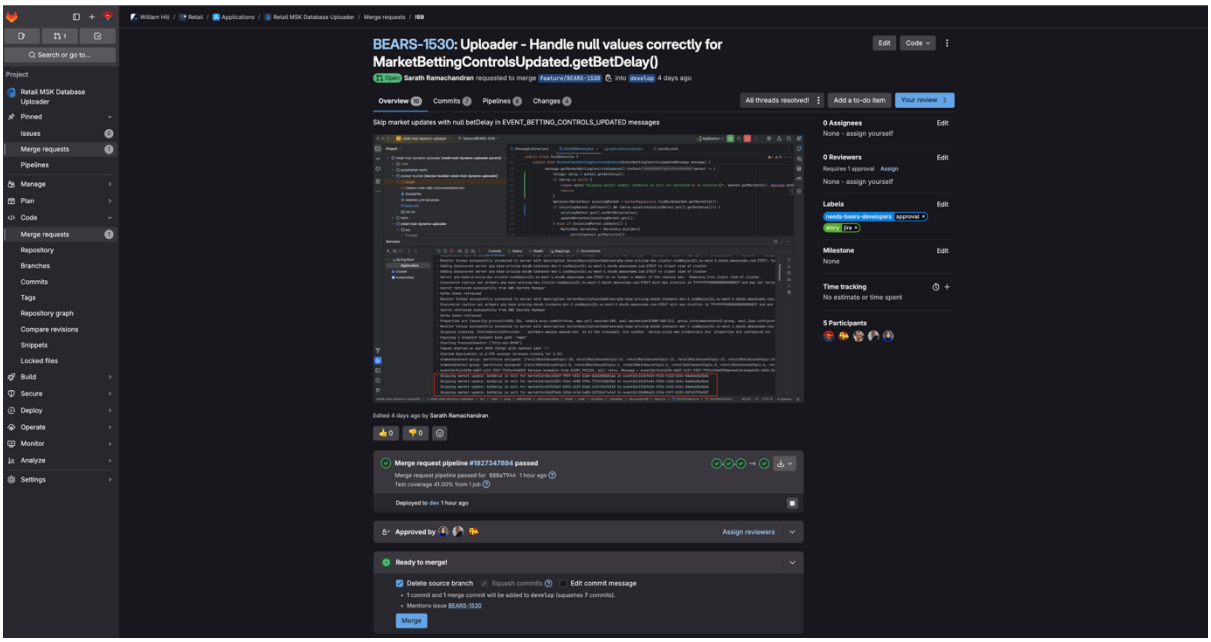

Figure 7 : MR overview

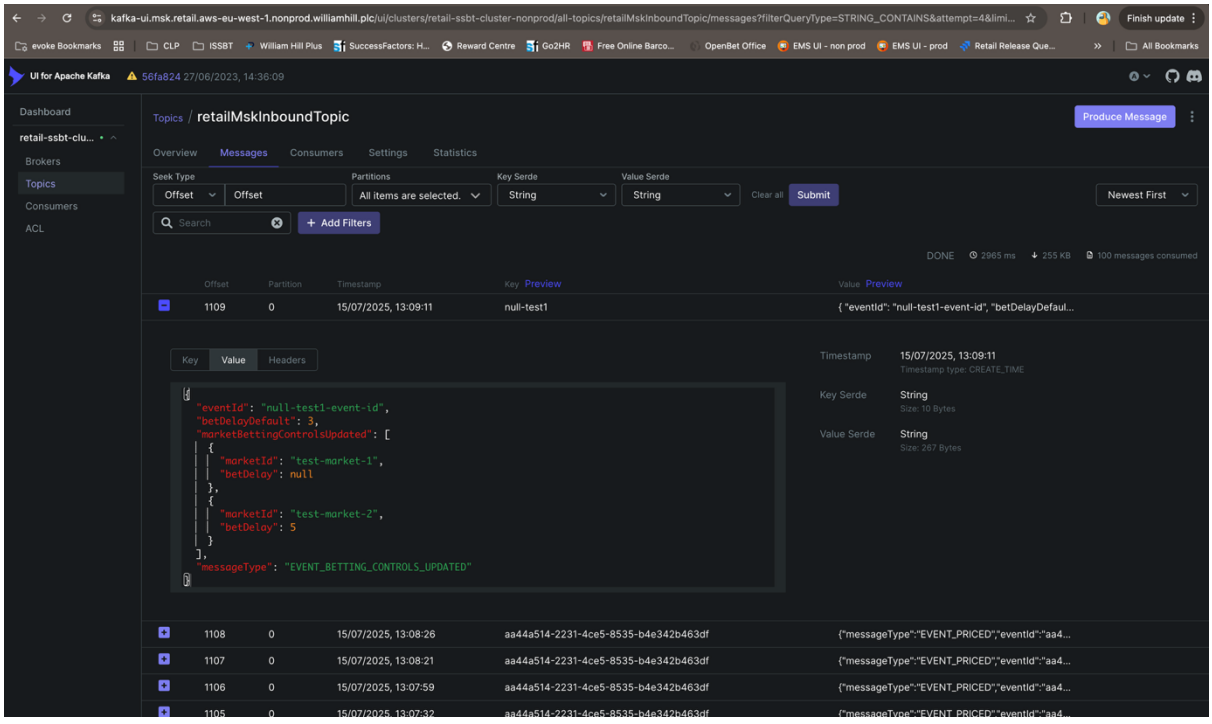

Figure 8 : Kafka sample message

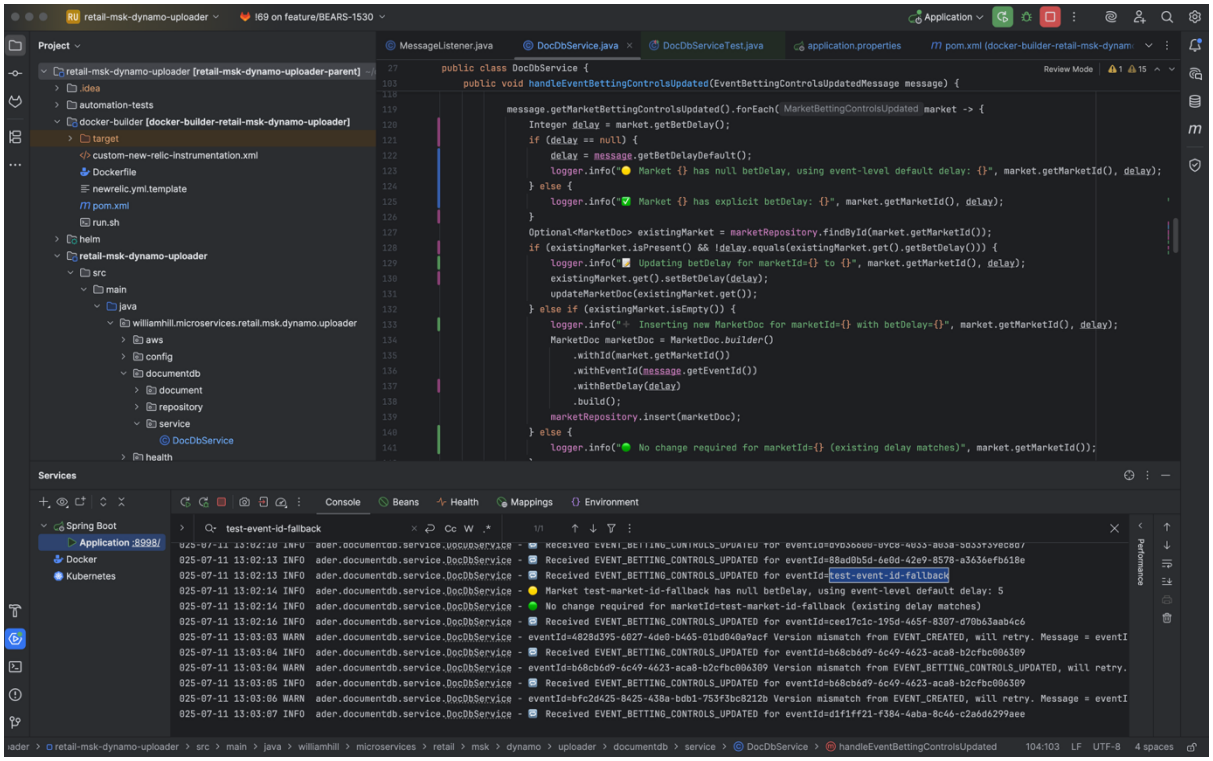

Figure 9 : Testing locally with Debugging logs

Communication and Collaboration

Throughout the BEARS-1530 implementation, I worked closely with my team to ensure alignment, quality, and smooth delivery:

- Daily Standups: Shared blockers and progress updates.
- Slack Conversations
- Merge Request Reviews

This collaborative process ensured cleaner code and a shared understanding across the team.

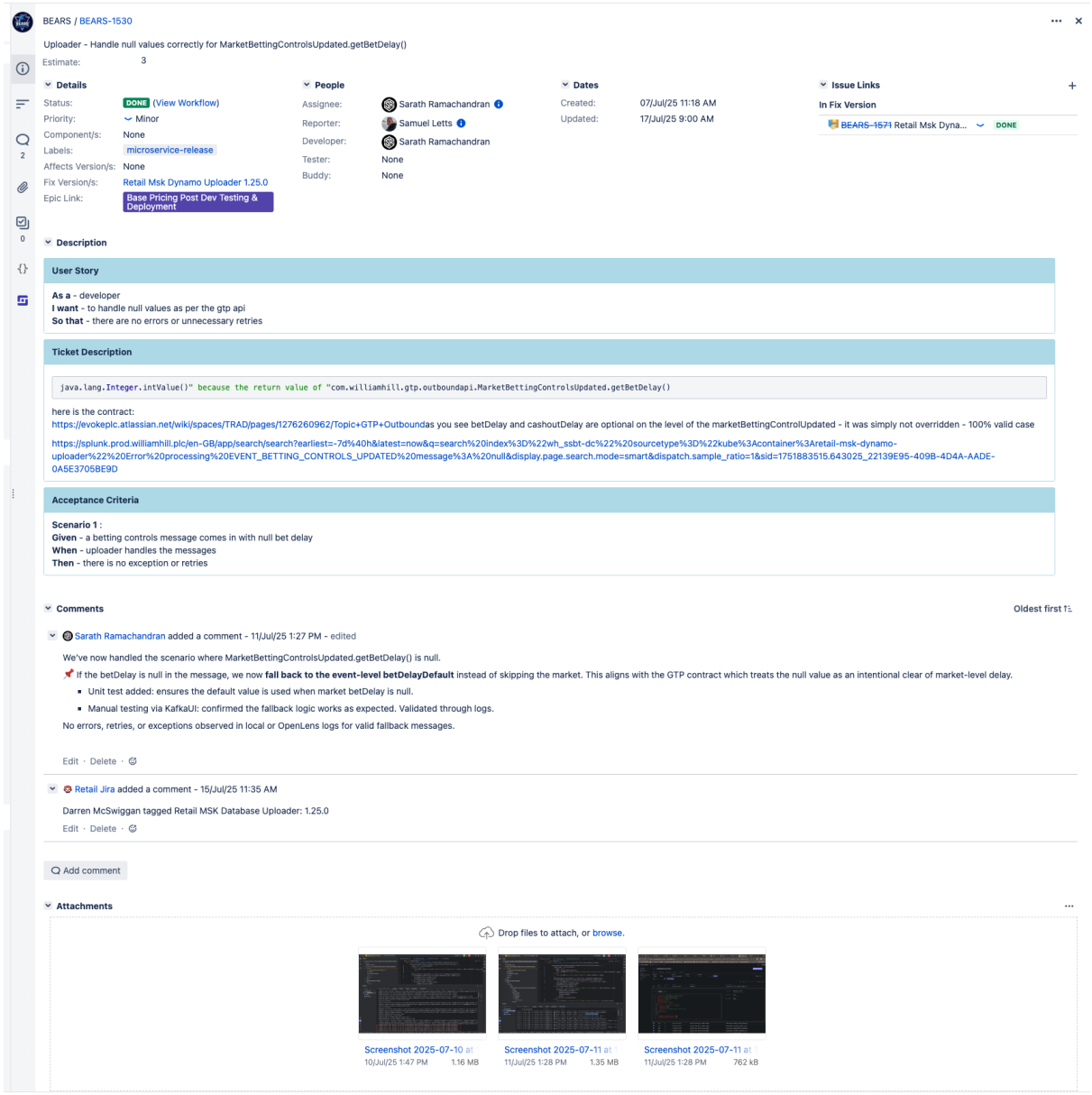

Figure 10 : Ticket status

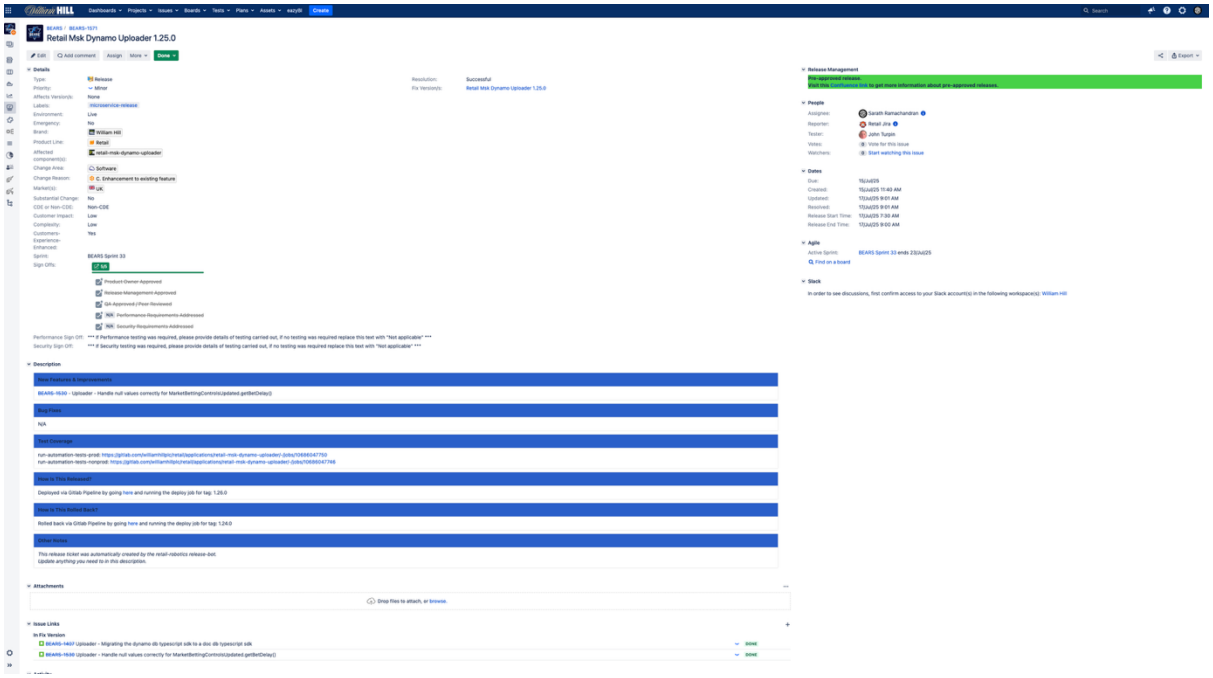

Figure 11 : Release Ticket status

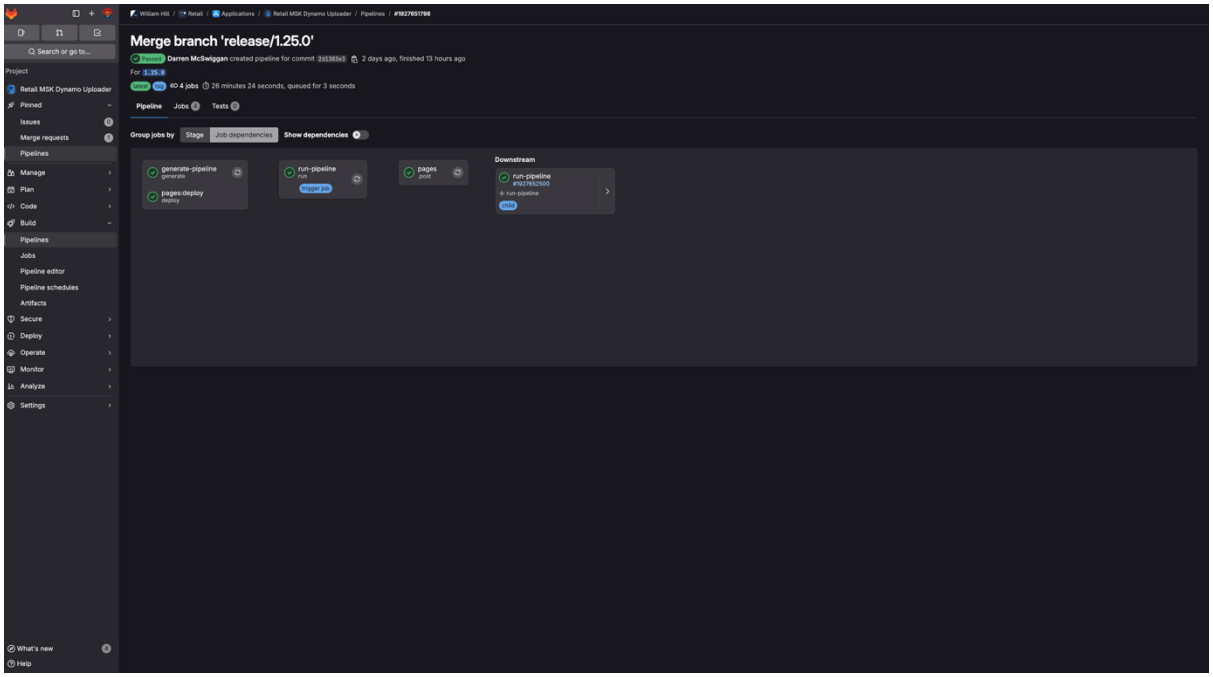

Figure 12 : Deployment pipelines

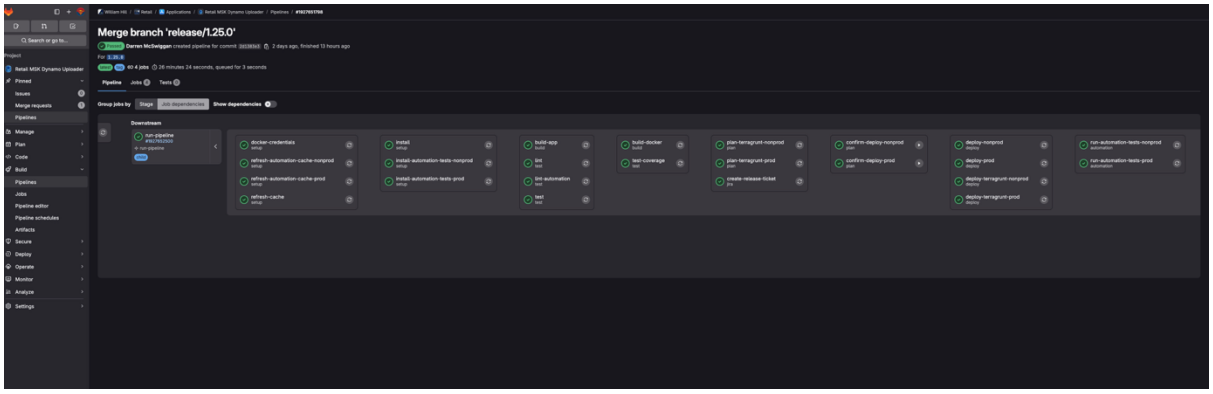

Figure 13 : Deployment pipelines

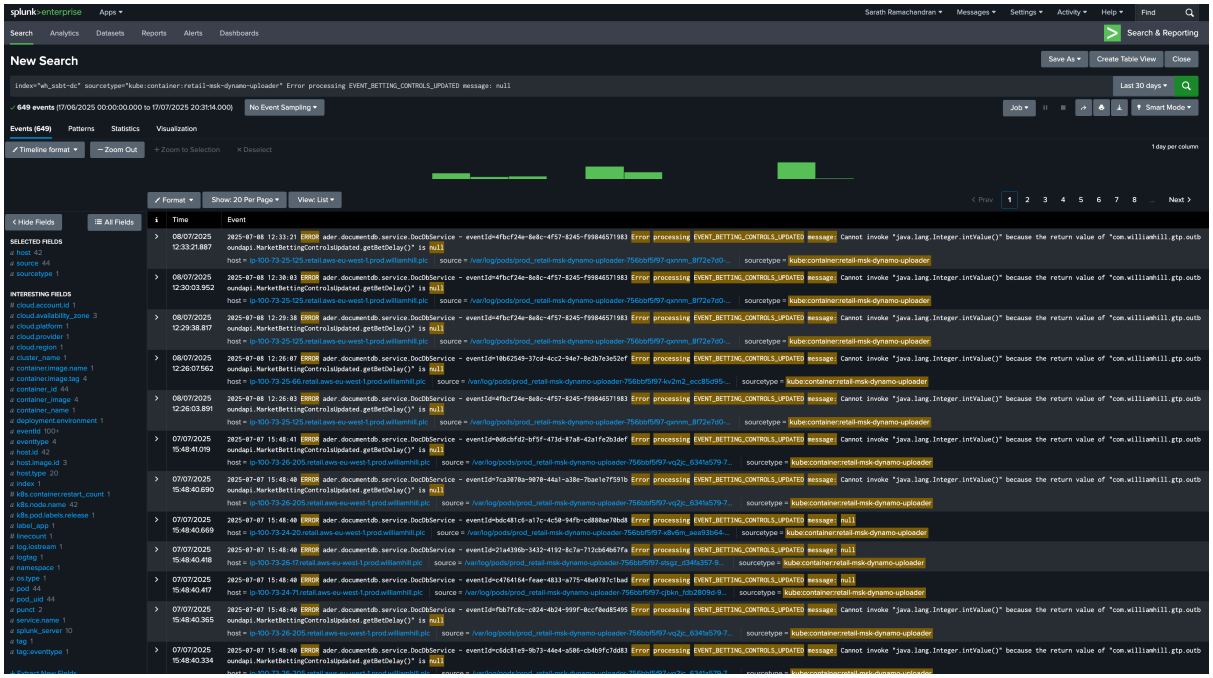

Figure 14 : Splunk logs (No error logs after deployment)

Result

The issue has been fixed with minimal and targeted code changes, a new fallback logic in place, and test coverage ensured. The feature is now live and behaving as expected.

Wins

- Identified and fixed a critical bug where null betDelay values were causing runtime exceptions.
- Implemented a clean fallback using betDelayDefault from the event message when market.betDelay was null, preventing failures and ensuring consistent behaviour.

- Added comprehensive unit tests to cover this fallback logic and validated functionality through automation tests.
- Coordinated effectively with automation testers and other stakeholders, including raising questions in Slack and clarifying edge cases with the data team.
- Successfully unblocked a failing pipeline by making an empty commit to bypass an ECR tag conflict.
- Delivered the feature with passing automation test runs and clean code in line with existing test standards.

What I Learned

- Gained hands-on experience with debugging null pointer issues in a live system.
- Learned how default event-level values (betDelayDefault) can serve as fallbacks and how to safely integrate them in message processing flows.
- Improved understanding of how DocDB inserts and updates work with Mongo repositories in Spring.
- Understood how compound Kafka messages like COMPOUND_MESSAGE are handled, and how our message processing logic differentiates between event-level and market-level updates.
- Learned how to identify and safely skip logs during noise-heavy conditions while still verifying changes using strategic logging.

Reflection

This ticket turned out to be more than a small bugfix — it involved understanding the core logic of how our event and market updates flow through message handlers, DocDB inserts, and Kafka topics.

Initially, it seemed like a logging cleanup task, but diving into the code and testing exposed a gap in handling missing bet delays. I appreciated how the team emphasized correctness and asked the right questions (e.g., “Should we fallback to default values?”). Collaborating with Sam, Bob, and the automation tester helped me understand the value of cross-checking assumptions, verifying default logic, and the power of clear, simple test cases.

This ticket was a good example of fixing a bug with long-term impact while keeping the system stable.